

ON VISIBILITY DETERMINATION IN SURFACES
WITH PARTIALLY ORDERED DATA

BY

SUE-LING CHEN WANG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1989

ACKNOWLEDGMENTS

I am deeply grateful to my advisor, supervisory committee chairman, and best teacher, Dr. John Staudhammer, for his invaluable inspiration, encouragement, advice, and help. Without his guidance and support, this work could never have been done. In addition to all the academic directions, he has had a significant influence on me in developing high principles and human strength in daily living.

I am also greatly indebted to all the other members of this committee, Graduate Research Professor Joseph Duffy of the Department of Mechanical Engineering and Professors Yuan-Chieh Chow, Douglas D. Dankel, II, Manuel E. Bermudez and Alexandros C. Papachristidis of the Department of Computer and Information Sciences, for their recommendations and advice regarding this dissertation. The responsibility for any remaining errors or shortcomings is, of course, mine.

I would like to thank sincerely all the members and good friends of the Computer Graphics Research Group, University of Florida, for their helpful discussions and suggestions. Special thanks are due to Professor Jiao-Ying Shi of Zhejiang University, Hangzhou for his assistance.

Financial support from the Departments of Computer and Information Sciences, Electrical Engineering, and Nuclear Engineering Sciences, College of Engineering, and Advanced Materials Research Center were very important to me and made the research and this manuscript possible. I am grateful to Dr. Larry L. Hench of the Department of Materials Science and Engineering for his contributions to my graduate career.

Finally, I wish to express my heart-felt gratitude to my parents-in-law and my mother for their never-ending encouragement and tremendous help. As for my husband, Shi-Ho,

words are insufficient to describe my feelings to him. Without his steady support, it would have been very difficult for me to finish this work. At last, but not least, I want to mention my beloved daughter, Jean, for her understanding and cheering while this work was being finished.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGMENTS.....	ii
ABSTRACT	vi
CHAPTERS	
1 INTRODUCTION	1
2 VISIBILITY DETERMINATION	8
Introduction	8
Literature Review.....	9
Rendering of Algebraic Surfaces	14
Data Generation and Manipulation	15
Published Algorithms	20
The Floating Perimeter Algorithm	22
Comparisons of Algorithms	31
3 SHADING	45
Introduction	45
A Proposed Shading Method	47
Diffuse Reflection and Ambient Light.....	57
Specular Reflection	58
Techniques of Shading Continuity	68
A Color-Ring Look-up Table.....	68
Display Considerations	72
4 ANTI-ALIASING.....	79
Introduction	79
Literature Review.....	83
Area Sampling Algorithms	88
Algorithm DISLOP_L for Line Rendering.....	89
Algorithm DISLOP_E for Edge Rendering.....	95
Algorithm DISLOP_C for Circle Rendering.....	99
Algorithm DISLOP_S for Sphere Rendering.....	102
Applications and Comparisons of Algorithms	108
Generation of Area/Intensity Sampling Look-up Tables.....	115
Post-anti-aliasing	125
5 EXTENSIONS OF THE ALGORITHMS	136
A Hidden-line Algorithm with Anti-aliasing/Post-anti-aliasing	136
Smoothed Visible Surfaces.....	144
6 CONCLUSIONS.....	156

BIBLIOGRAPHY	159
BIOGRAPHICAL SKETCH	167

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ON VISIBILITY DETERMINATION IN SURFACES
WITH PARTIALLY ORDERED DATA

By

SUE-LING CHEN WANG

August 1989

Chairman: Dr. John Staudhammer

Major Department: Computer and Information Sciences

Hidden-line/surface elimination is very important in computer graphics, because it makes the realistic rendering of three-dimensional objects possible. For special classes of objects, simplified algorithms are often proposed for utilizing geometric properties and projections of the objects. An example is the class of solid objects defined by partially ordered surface data, such as objects of algebraic surfaces. A simple algorithm, the Floating Perimeter Algorithm, has been developed for rapidly determining the visibility of such surfaces on a raster display from any chosen viewing direction. This algorithm is mainly based on two procedures: the enumeration of facets and the formation of a floating perimeter for the grid surface representation of an algebraic surface.

Shading is also one of the requirements to produce pleasing computer-synthetic images. Fast Phong shading technique employs the two-dimensional Taylor series and forward differencing with the Torrance-Sparrow specular reflection model to achieve computational efficiency. However, the original Phong model, used in this dissertation, may be made computationally just as efficient using the procedures developed here.

Comparisons between the original and the modified procedures are made by analysis of algorithms and juxtaposition of rendered images. In addition to the three basic shading techniques--constant shading; intensity interpolation, or Gouraud shading; and normal-vector interpolation, or Phong shading--another approach is presented for quickly rendering surfaces given by complex variable functions. The technique of shading continuity is to represent each phase angle of the function surface by a color scale in a color-ring look-up table with the closure property of a circle, in which angles can be measured smoothly: at the completion of a full circle one should be back to the same color.

When the quality of an image is a concern, its annoyingly jagged silhouettes and edges need to be dealt with. To eliminate these aliasing artifacts, area sampling algorithms have been well-developed to efficiently render raster images of lines, polygon edges, circles and sphere silhouettes. The anti-aliasing line algorithm DISLOP_L can produce constant-intensity smooth lines, whereas some other published algorithms do not. Also, these algorithms can be used to smooth any kind of object image. For more rapid smoothing of images, one must precalculate area/intensity sampling look-up tables from these algorithms for post-anti-aliasing.

Combining algorithms of visibility determination, fast Phong shading, and anti-aliasing in different ways, extended algorithms are created for a hidden-line algorithm with anti-aliasing/post-anti-aliasing and a smoothed visible surface algorithm for the rapid generation of smoothed realistic images of objects defined by partially ordered surface data.

CHAPTER 1 INTRODUCTION

Computer graphics is used to generate images in a variety of fields, including CAD/CAM, scientific visualization, medicine, simulation, animation and art [Ward89]. The vast majority of these visualizations use a raster display. For instance, image rendering of real objects on a raster display may create synthetic images indistinguishable from photographs of the objects in the real world. This kind of image synthesis is a subfield of computer graphics that attempts to produce photo-realistic pictures. Another subfield can be the making of pictures of abstract objects for visual representation of mathematical equations.

The first step in rendering images of three-dimensional opaque objects is the choice of a view direction that decides not only which parts of the object are visible and thus must appear on the picture, but also which parts are concealed and thus must be omitted. This task of visibility determination is known as hidden-line or hidden-surface elimination, depending on whether the object image is drawn with just simple lines or shaded surfaces, respectively. Many algorithms [Suth74, Fole82] have been developed and presented for determining visible parts of the opaque objects in the physical space as well as in the screen coordinate system. These algorithms are classified as two categories, object-space and image-space, and in general, their performance characteristics are difficult to compare since they are substantially different.

For special classes of opaque objects, algorithms of visibility determination can utilize geometric properties of the objects and their projections, and thus often become simpler and faster. An example is the class of solid objects defined by partially ordered surface data, such as objects of algebraic surfaces. The Floating Horizon Algorithm (abbreviated the

FHA) [Will72, Wrig73, Roge85] is one of the most frequently used methods for removing hidden lines from three-dimensional representations of algebraic surface functions. The algorithm uses a series of parallel cutting planes at constant values of x , y , or z to intersect an algebraic surface and thus converts the 3-D problem to a series of 2-D ones. However, only a set of X-, Y-, or Z-direction lines, instead of two sets of orthogonal curves, are drawn in the grid that represents the object surface.

The hidden-line elimination algorithm proposed by Anderson [Ande82] uses a grid surface consisting of a set of grid elements to approximate the algebraic surface. Therefore, the algorithm does not have restrictions of the FHA. In addition, since grid surfaces and their projections have geometric properties which permit the elimination of hidden lines to be done more easily than in the general case, the algorithm is fairly simple and fast. Nevertheless, it has to treat some particular viewing cases as special ones which, in turn, increases its computational complexity.

A more general solution to the imaging problem for the solid objects defined by partially ordered surface data has been developed and described in detail in Chapter 2. It is called the Floating Perimeter Algorithm [Wang85a, Wang86], an image-space algorithm that takes advantage of the object coherence and the limitation of a raster display to establish a visible changing periphery for determining the visibility of each point rendering the object surface. The algorithm also uses grid surface representations but with more flexibility in data generation and manipulation--by selecting a set of uniform or non-uniform grid elements for constituting the surface. A non-uniform grid surface has "reasonable" non-uniform intervals between adjacent grid points, but the process is seldom used. Compared to uniform grid spacing, it can use fewer or the same number of grid elements to represent the same object in the same or more detail, and, therefore, requires less or the same amount of computation and display.

The Floating Perimeter Algorithm (abbreviated the FPA) has been compared with the FHA and Anderson's algorithm. The results show that its execution time appears to be a bit

better than the others. It has unique properties, such as generality and flexibility. It does not have restrictions of the FHA and treats special cases in Anderson's algorithm like the others. Image size is a factor of execution time as well as the number of facets, and non-uniform grid surface approximation can be chosen as well as a uniform one. In other words, the FPA can efficiently and exactly determine the visibility of three-dimensional solid objects defined by partially ordered surface data in order to produce a realistic image on a raster display.

After the first part of the rendering problem, visibility determination, is solved, a computer-synthesized image of opaque objects with only visible parts will be displayed on a raster scan device. Hidden-line or hidden-surface elimination that has just been solved: a raster image of opaque objects can be represented by visible lines or visible surfaces, respectively. To display a visible-line image of objects, the depth of an object can be represented by varying degrees of intensities: objects intended to appear closer to the viewer are displayed at higher intensity. For implementing intensity depth cueing, depth (z-coordinate) information must be provided. Otherwise, the image is displayed with constant intensity lines.

On the other hand, when displaying a visible-surface image of objects on a raster device, we have to consider the intensity distributions on the objects and then the intensity of each pixel. These tasks are called "shading techniques." Three basic shading techniques - constant shading; intensity interpolation, or Gouraud shading [Gour71]; and normal-vector interpolation, or Phong shading [BuiT75]-- are commonly adopted. Phong shading requires the most computation but does remedy many shading problems of the other two techniques. Of the three it produces the best shading approximation and is most popularly applied.

For reducing the computational complexity, a fast Phong shading technique is presented in a work by Bishop and Weimer [Bish86]. It uses two dimensional form of Taylor series to approximate the reflection equation and then combines this with the

forward differences technique. For handling Phong's reflection model with an equation considering ambient, diffuse and specular reflectances, this method can evaluate the intensity per pixel with only 5 additions and 1 memory access. The specular reflection model in this method actually employs the Torrance-Sparrow model [Torr67], which was first applied to computer graphics by Blinn [Blin77]. Therefore, this fast Phong shading technique has been modified to use the original Phong's specular reflection model and described in Chapter 3. The differences between these two approaches are discussed in detail and photographs of the resulting screen images are compared.

In addition to the shading techniques described above, some other properties can be applied for smoothly shading images of objects defined by partially ordered surface data. An example is to render a surface given by a complex variable function in accordance with its phase angle by specifying a particular color look-up table [Stau75, Stau78]. This technique has been improved [Wang85b]. That is, the total number of colors in the coloring look-up table is increased to be twice the original number, and the method of averaging phase angles of each triangle for shading continuity of the function surface is modified. Also, the removal of hidden surfaces can be done by applying steps (I) to (III) of the Floating Perimeter Algorithm and then displaying only visible parts of the enumerated facets in the front-to-back order with flag checking in the floating perimeter. When Painter's algorithm is applied, all the enumerated facets are displayed in the back-to-front order and no flags need to be considered.

Therefore, a very smooth shaded surface image of multi-dimensional function can be displayed by using the Ring color look-up table and the Floating Perimeter Algorithm or Painter's algorithm. Besides, an interactive technique of replacing any color in the look-up table had also been developed to identify any given phase angle without additional computation. In addition, a sequence of functional surfaces was also generated on a Run-Length Decoder [Chen83] for rotation in real time so that one can visualize an object from

any direction and understand it more easily. Chapter 3 details these techniques and illustrates the results.

The raster images of objects represented by visible lines or visible shading surfaces often show annoyingly jagged appearances on their edges. This occurs because the images on the resolution-limited raster display are discrete representations of continuous objects in geometric space and thus have aliasing artifacts. Therefore, problems associated with raster scan display techniques are of concern to those who are interested in high quality three-dimensional image generation. Applications, such as molecular modeling, movie animation, even text display and mathematical modeling, are all affected by the aliasing problems.

Of all anti-aliasing techniques, the solution proposed by Fujimoto and Iwata [Fuji83] is one of the most rapid and simple methods. Their algorithm shows that each pixel intensity is inversely proportional to the distance from the vector's center line and uses an incremental calculation to generate both the pixel's intensity and its position. However, it also shows that the calculation of the intensity for a unit length of the vector does not account for any given vector slope and thus the vector produced can not have the property of constant-intensity: the intensity per unit length is not constant.

Some discrete area sampling techniques for removing the aliasing artifacts from raster images of lines, polygon edges, circles and sphere silhouettes [Wang84] have been well-developed recently. They all utilize the symmetry properties of objects and floating point operations to precisely calculate the pixels' area coverages for modulating the light intensity in order to anti-alias the objects exactly. Techniques for smoothing silhouettes of circles and spheres actually utilize those for anti-aliasing lines and polygon edges, respectively. An anti-aliased line is produced according to its slope and end points of floating point coordinates and thus has the property of constant-intensity. In fact, the calculations of these algorithms are based on two factors, the vector slope m and the horizontal/vertical distance

dxy, and can be implemented for smoothing raster images of any kinds of objects, including molecular models and objects represented by visible lines or shaded surfaces.

The area sampling algorithms for anti-aliasing lines and polygon edges are also utilized to generate area/intensity sampling look-up tables. The table is a three-dimensional array: the first index is for various slopes m_i , the second index is for different distances dxy_j , and the third one is the maximum number of pixels passed by a line or an edge in one column/row. The dimension of the table is chosen to be $51 \times 51 \times 3$ and its contents can be either floating point area coverages or integer intensity values. Thus, all these floating point/integer values in the area/intensity sampling look-up tables $A[51][51][3]$ and $I[51][51][3]$ respectively need to be calculated from the above anti-aliasing algorithms and then stored for use in the rapid post process of anti-aliasing images, i.e. post-anti-aliasing images. The size of the tables may be changed. Also, the area/intensity sampling look-up tables could be used to make anti-aliasing post-processors for providing fast and proper anti-aliasing features on raster display devices.

Chapter 4 describes the origin of an alias, the sampling theorem [Oppe75], various kinds of aliasing effects that occur in computer graphics, published anti-aliasing algorithms for removing these effects, some well-developed area sampling algorithms, the generation of area/intensity sampling look-up tables, and post-anti-aliasing.

Algorithms of visibility determination, shading, and anti-aliasing can be combined to create extended algorithms for the image generation of objects given by partially ordered surface data with hidden-line/surface elimination and anti-aliasing/post-anti-aliasing. That is, visible smoothed lines or shaded surfaces and smoothed silhouettes are used to represent the object images. Since the difference between images that are anti-aliased or post-anti-aliased is very little, almost indistinguishable, computation time is the main concern. Therefore, utilizing the area/intensity sampling look-up tables for post-anti-aliasing can produce a relatively high quality image while reducing computational complexity. These algorithms are discussed in Chapter 5.

Finally, Chapter 6 summarizes the results and discusses further work that can be done in the near future. In summary, the major contributions of this work are the design of various algorithms of visibility determination, fast Phong shading, anti-aliasing and their extensions for the rapid generation of smoothed realistic images of solid objects defined by partially ordered surface data. It is worth noting that all these algorithms are very efficient and fairly simple.

CHAPTER 2 VISIBILITY DETERMINATION

Introduction

One of advantages of computer graphics is that one can create synthetic images of an object for visualization without having to physically construct the object. While it is relatively easy to produce a perspective picture of a transparent object made up only of lines, it is somewhat more difficult to produce a realistic rendering of an opaque object. The opaque object is more difficult to show because one must decide not only where each part of the object will appear in the picture, but also whether to show any part at all. Some parts of an opaque object will be concealed in any view of it; a computer programmed to make pictures of opaque objects must be able to determine which parts are visible in the chosen view and thus must be shown, and which parts are hidden and thus must be omitted.

The task of visibility determination of an object on a vector display was originally known as the "Hidden-Line Problem," because it amounted to finding and eliminating--or making dashed--all of the lines in an output drawing which were hidden. When the shaded pictures were being produced by computer on the raster display, a variant of the problem, the "Hidden-Surface Problem," became important. In a shaded picture one must include or omit surface areas rather than just the lines representing edges.

The hidden-line and hidden-surface problems in computer graphics have been discussed for more than two decades. Many solutions and algorithms have been produced and developed. Based on the coordinate system or space in which the algorithms are implemented, they can be classified as object-space or image-space. Object-space algorithms are implemented in the physical coordinate system in which the objects are described and thus are also called continuous algorithms. The geometrical relationships

among the objects in the scene are calculated in order to determine which parts of which objects are visible. These results are generally accurate to the precision of the machine and can be satisfactorily enlarged many times. Image-space algorithms are implemented in the screen coordinate system in which the objects are viewed. The visibility of the final image is decided point by point at each pixel position on the display device. That is, calculations are performed only to the precision of the screen representation. Therefore, image-space algorithms are also called point-sampling algorithms. Most hidden-surface algorithms use image-space methods, but object-space methods can be used effectively in some cases. Hidden-line algorithms generally employ object-space methods, although many image-space hidden-surface algorithms can be readily adapted to hidden-line determination.

These two approaches generally lead to various hidden-line and hidden-surface algorithms with different performance characteristics. Theoretically, the computational work for an object-space algorithm that compares each object in a scene with the remaining $(n-1)$ objects grows as the number of objects squared (n^2). Similarly, the work for an image-space algorithm which compares every object in the scene with every pixel location in screen coordinates is proportional to nN , where N is the number of pixels, typically (512^2) or more. While one might therefore believe that object-space algorithms require less work than image-space algorithms even when n exceeds 100,000, in practice, this is not the case. Most of the individual steps in object-space algorithms are more time-consuming while it is easier to take advantage of coherence in raster scan implementation of image space algorithms.

Literature Review

Roberts [Robe63] devised the first known solution to the hidden-line problem. His algorithm is implemented in the object space and restricted to the convex objects. It tests each relevant edge to see if it is obstructed by the volume occupied by some objects in the environment and thus capitalizes on the spatial coherence of objects. The computation

required by this algorithm grows roughly as the square of the number of objects in the scene: each edge of a body must be tested against every object in the scene.

Edge-intersection algorithms [App67, Lout67, Gali69] are also object-space algorithms, but they are more concerned with edge intersection than with face relationships. They use "edge coherence" to locate and sort all intersections along a relevant edge in order to establish the quantitative invisibilities of all points on the edge. However, a number of unpleasant effects, such as overlapping of images of edges and image of a vertex lying on the image of another edge, can occur which require careful attention for correctly computing the invisibility.

These limitations of previous object-space hidden-line algorithms have thus prompted the development of more general solutions [Suth74, Hedg82]. A software package for displaying a three-dimensional solid object modeled by polygons by using a few fundamental hidden-line algorithms has also been presented [Crow81b, Crow81c]. Recently, Blinn presented a method called Fractional Invisibility [Blin88] that he used to solve the hidden-line problems which occurred during animation of a scene about human evolution for the Cosmos television series in 1980. The problems occur when two distinct objects happen to line up in such a manner that a vertex of one object projects exactly on an edge of the other. To handle these situations, Blinn carefully assigned a fractional invisibility increment to each edge for determining the visibilities of all its intersected segments. This solution worked well enough to get the evolution scene done. However, in the unusual case where a point accidentally coincides with another unrelated point, the comparison of face-vertex is still a problem.

The painter's algorithm [Schu69, Newe72] is one of the earliest visible-surface algorithms developed. This algorithm is based on an analogy to painting with opaque paint, where closer objects are painted over farther ones. In computer graphics, objects are first sorted with respect to distance from the viewpoint and then are painted sequentially into a frame buffer, or an image space. A serious drawback of this algorithm is that a global order

for drawing the objects needs to be established. This ordering could be very expensive since it involves every object in the scene. Testing for cyclic overlaps between objects is also very expensive. Therefore, the painter's algorithm is generally not used for complex scenes unless some assumptions allow simplifications in, or elimination of, the sorting and overlap testing. However, modeling systems based upon small particles that are suitable for rendering with the painter's algorithm have been developed [Reev83, Reev85]. Since the particles are small, there is little chance for cyclic overlaps. The papers present stochastic algorithms for generation of particles and approximate algorithms for shading and visible-surface determination using the painter's algorithm.

The Scan-line algorithm [Wyli67, Bouk69, Watk70], one of the most commonly adopted image-space algorithms, creates a synthetic image one scan line at a time and usually processes scan lines from the top to the bottom of a raster display. As each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. At each pixel position along the scan line, depth calculations are made for each intersecting surface to determine which is nearest to the view plane. Thus, scan-line coherence and point-to-point coherence along the scan line are utilized in this algorithm.

Another image-space approach, called z-buffer or depth-buffer algorithm [Catm74, Catm75], requires not only a frame buffer in which intensity values are stored, but also a depth-buffer in which z-values can be stored for each pixel. Surfaces are compared for each pixel at a time. The nearest surface at that pixel is visible and its intensity value at that position is written to the frame buffer. This algorithm is very simple to implement, but it requires a large amount of space for the z buffer. The A-buffer (anti-aliased, area-averaged, accumulation buffer) algorithm, proposed by Carpenter [Carp84] as an enhancement to the Z-buffer method, is a general hidden surface mechanism suited to medium scale virtual memory computers. It uses a box filter, a 4x8 bit mask representing the subpixel polygons, and implements Boolean operations to resolve visibility among an arbitrary collection of

opaque, transparent and intersecting objects at each pixel, and thus effectively increases the image resolution of the Z-buffer algorithm several times.

A special case of the z-buffer algorithm, the scan-line z-buffer algorithm proposed by Myers [Myer75], uses a display window only one scan line high with a width of the horizontal resolution of the display. Therefore, the scan line frame buffer and its z-buffer need only be 1 bit high, the horizontal resolution of the display wide, and the requisite precision deep. Another scan-line z-buffer algorithm, presented by Dyer and Whitman [Dyer87] for implementation on vector computers, takes advantage of the architecture of a vector computer and thus renders a scene by employing vectorization for improved performance.

Scan-line algorithms for curved surfaces have also been developed [Blin78, Whit78, Lane80, Rock87] for displaying parametric bipolynomial, typically bicubic or Bezier, surfaces directly from the surface description in scan line order. Curved surface representations are important in computer graphics applications, because they are often more compact than polygonal representations, and hence require relatively compact databases. A modified spanning scan-line algorithm [Athe83], which integrates knowledge of a Boolean construction tree in the surface resolution process, displays visible surfaces of constructive solid geometry models and demonstrates significant efficiency improvements over previous scan-line hidden-surface removal approaches. Another technique called invisibility coherence [Croc84] is also proposed to remove portions of a scene that are unlikely to be visible and to decrease the time necessary to render shaded images by the existing scan-line hidden-surface algorithms.

The scan-plane-sweep algorithm, proposed by Hamlin and Gear [Haml77], is very similar to the scan-line visible-surface algorithms, but requires less depth calculation. As in the scan-line algorithm, a continuously moving scan plane stops only at positions where the visibility could change, instead of at equally spaced scan lines. Nurmi [Nurm85] has shown that the scan-plane-sweep algorithm can be implemented using time and space

proportional to $(n+k)\log n$, where n is the number of edges in the input data and k is the number of edge intersections in the picture plane. For a worst case picture, the number k of intersections can be proportional to n^2 , and since all these intersections are visible, n^2 is also the complexity of the output. McKenna [McKe87] has presented a similar algorithm, which takes time and space proportional to n^2 and is, therefore, optimal for the worst case with a maximal number of visible-edge intersections.

Using area coherence based on recursive subdivision of the image area into quarters, the area-subdivision algorithm, developed by Warnock [Warn69], follows the "divide and conquer" strategy. At each stage, the polygons overlapping the current area are compared to see if there is a "blocking" polygon that surrounds the area and is in front of all other polygons, or if only a single polygon overlaps the area. In these simple cases, the single visible polygon can be rendered directly into the area. Otherwise, the area is divided into quarters and each quarter is treated recursively. Another strategy, presented by Weiler and Atherton [Weil77], subdivides the screen area along polygon boundaries rather than along rectangle boundaries. Therefore, its result is a set of visible polygons making up the scene, and it can reduce the number of subdivision steps from that required by the previous method.

Based on the use of hierarchical data structures such as a 2-D image-space quadtree, octree algorithms have also been developed [Frie85, Same88a, Same88b] to accomplish hidden surface elimination. An octree encoding scheme divides regions of three-dimensional space into octants and stores eight data elements in each node of the tree. When an octree representation is used for the viewing volume and the octree nodes are projected onto the viewing plane in a front-to-back order, objects in octants 0, 1, 2, 3 obscure objects in the back octants (4, 5, 6, 7). Since this algorithm stores the entire solid region of an object, it is generally applied to medical imaging for generation of cross-sectional slice of solids, and is also applied to solid modeling for generation of more

complex solid objects using variants of Boolean set operations such as union, intersection, and set-difference for such primitives as cubes, parallelepipeds, cylinders, and spheres.

Although many algorithms on visibility determination are briefly reviewed and described in the above, a taxonomy of algorithms given by Grant [Gran87] and a tutorial and reprint collection on image synthesis presented by Joy et al. [Joy88] are more complete. It is noted that the hidden-line and hidden-surface problems are often simpler when restricted to special classes of objects. An example is the class of solid objects defined by partially ordered surface data, such as objects of algebraic surfaces. Many algorithms for determination the visibility of such objects have been developed and are discussed in the following section.

Rendering of Algebraic Surfaces

The Mathematical description of three-dimensional surfaces usually falls into one of two classifications: parametric and implicit. An implicit surface is defined to be all points which satisfy some equation $f(x,y,z)=0$ and is usually called an algebraic surface. Algebraic surfaces arise from diverse applications in mathematics, engineering, and sciences as well as other disciplines. In computer aided geometric design and graphics, degree one and degree two algebraic surfaces are widely used. For instance, quadric surfaces (degree two), including such shapes as ellipsoids, spheres, cylinders, cones, paraboloid, hyperboloids of revolution and so forth, are often used for representing elements and parts of many geometric objects. Therefore, through computation and graphical display, these objects can be visualized more precisely and efficiently.

Defining by implicit equations, algebraic surfaces are also solid objects defined by partially ordered surface data. Geometric properties of these surfaces allow rendering of this special class of objects to be done more easily than in the general case. In the following discussion, specific attention is paid to single z-valued algebraic surfaces, although some ideas presented can be applied to any algebraic surface and some parametric surfaces.

Data Generation and Manipulation

For any single valued algebraic surface with bivariate function of the form:

$$z = f(x, y)$$

we define an algebraic surface over the rectangle $(X_{\min}, X_{\max}, Y_{\min}, Y_{\max})$ in the World Coordinate System, i.e., in the 3-D space with Cartesian Coordinates. Then we cut the surface with N_x parallel planes at constant values of x and with N_y parallel planes at constant values of y . The spacing between these planes are:

$$D_x = (X_{\max} - X_{\min}) / (N_x - 1)$$

$$D_y = (Y_{\max} - Y_{\min}) / (N_y - 1).$$

The number of grid points is $N_x * N_y$. Their locations (x_i, y_j) and corresponding z , i.e. z_{ij} , values are computed as follows:

$$x_i = X_{\min} + D_x * (i - 1)$$

$$y_j = Y_{\min} + D_y * (j - 1)$$

$$z_{ij} = f(x_i, y_j)$$

where $i = 1, \dots, N_x$, and $j = 1, \dots, N_y$. By connecting adjacent grid points for $i = \text{constant}$ and $j = \text{constant}$ with straight lines, the algebraic surface is approximated by a grid surface consisting of $(N_x - 1)(N_y - 1)$ straight-edged grid elements. The image of a grid element of the function is called a facet. Therefore, the image of a grid surface is composed of a set of facets on the projection plane.

Instead of generating data through computation, a database of solid objects with partially ordered surface data can be directly taken from an ordered list. An example is shown in Table 2-1. The defined limits of the surface in the X- and Y-direction-- X_{\min} , X_{\max} , Y_{\min} , and Y_{\max} -- are specified first. Two integers, N_x and N_y , then follow. N_x and N_y indicate the number of planes for cutting the algebraic surface and parallel to the Y-Z and X-Z planes, respectively. Finally, a set of $N_x * N_y$ triplets in real numbers represent the world coordinates (x_i, y_j, z_{ij}) of grid points on the surface in the object space.

Table 2-1. A database of an object, which is an algebraic surface with a bivariate function $z = \exp(-x^2 - y^2)$ defined over the rectangle $(-1.0, 1.0, -1.0, 1.0)$ in the World Coordinate System.

A DATABASE	VARIABLES' NAMES
-1.0 1.0 -1.0 1.0	$X_{\min}, X_{\max}, Y_{\min}, Y_{\max}$
11 11	N_x, N_y
-1.0 -1.0 0.1353	(x_1, y_1, z_{11})
.....
1.0 -1.0 0.1353	(x_{N_x}, y_1, z_{N_x1})
-1.0 -0.8 0.1940	(x_1, y_2, z_{12})
.....
1.0 -0.8 0.1940	(x_{N_x}, y_2, z_{N_x2})
.....
-1.0 1.0 0.1353	(x_1, y_{N_y}, z_{1N_y})
.....
1.0 1.0 0.1353	$(x_{N_x}, y_{N_y}, z_{N_xN_y})$

Whether computed or given in databases, the algebraic surfaces are defined as solid objects with partially ordered surface data and are approximated by grid surfaces with uniform intervals between adjacent grid points in the X- and Y-directions. However, all intervals between each two adjacent grid points do not need to be constant. In other words, the spacing between parallel planes in the X- and/or Y-directions can be non-uniform, which allows an algebraic surface be approximated by a grid surface with grid elements of different sizes. The non-uniform grid surface representations of algebraic surfaces have some advantages and disadvantages. The advantages include using fewer or the same number of grid elements to represent the same object in the same or more detail, and, therefore, requiring less or the same amount of computation and display than those that are drawn with uniform grid surfaces; the main disadvantage is just that the visualization of non-uniform grid surfaces is a little unusual.

The techniques for generating computed data of grid points with non-uniform spacing are more complex. Here, we describe a fairly simple way:

On an algebraic surface defined over the rectangle $(X_{\min}, X_{\max}, Y_{\min}, Y_{\max})$ in the World Coordinate System, we temporarily slice the surface with maximum $N_{\max x}$ parallel planes at constant values of x and with maximum $N_{\max y}$ parallel planes at constant values of y . Therefore, the minimum step sizes between these planes are:

$$D_{\min x} = (X_{\max} - X_{\min}) / (N_{\max x} - 1)$$

$$D_{\min y} = (Y_{\max} - Y_{\min}) / (N_{\max y} - 1)$$

which indicate the minimum distances between the temporary adjacent planes paralleled to the Y-Z and X-Z planes, respectively. The total number of temporary grid points is $N_{\max x} * N_{\max y}$. The world coordinates of these grid points (x_a, y_b, z_{ab}) are then computed as follows:

$$x_a = X_{\min} + D_{\min x} * (a - 1)$$

$$y_b = Y_{\min} + D_{\min y} * (b - 1)$$

$$z_{ab} = f(x_a, y_b)$$

where $a = 1, \dots, N_{\max x}$, and $b = 1, \dots, N_{\max y}$. In addition to these two minimum step sizes, there are two other maximum step sizes, $D_{\max x}$ and $D_{\max y}$, which should be given as some multiples of $D_{\min x}$ and $D_{\min y}$, respectively. In other words, ratios of the maximum to minimum step sizes in the X- and Y-directions, R_x and R_y , need to be specified as integers. Hence, the relationships between all step sizes and ratios are

$$D_{\max x} = D_{\min x} * R_x \quad \text{and}$$

$$D_{\max y} = D_{\min y} * R_y.$$

$D_{\max x}$ and $D_{\max y}$ also indicate the maximum distances between the final adjacent parallel planes for cutting the algebraic surface in the directions of the Y-Z and X-Z planes, respectively. In order to represent the surface with non-uniform intervals, an important value, Z_t , called the Z tolerance, needs to be given for comparison with the sum of differences of z values between adjacent grid points. Since Z_t can be changed at the user's ease, one can use a large number of test cases to determine its best value in order to produce a more-detailed and good-looking non-uniform grid surface with the same amount or less computations.

Two one-dimensional arrays XA and YA are created to store X and Y values of all needed cutting planes for approximating the algebraic surface by an appropriate grid surface with "reasonable" non-uniform intervals between adjacent grid points. The array XA is initialized with X values of two limits X_{\min} , X_{\max} and the array YA with Y_{\min} , Y_{\max} . Then, starting at the grid point (x_2, y_2) , its function value z_{22} is compared with function values of two previous adjacent grid points in the X-direction, z_{12} , and in the Y-direction, z_{21} , to see if the sum of two z differences changes by more than the amount Z_t in the Z-direction. If it is, x-coordinates of these three grid points are stored in the array XA and y-coordinates in the array YA. Similarly, each function value, z_{ab} , of grid points of several series $(x_3, y_2), \dots, (x_{N_{\max x}-1}, y_2), (x_2, y_3), \dots, (x_{N_{\max x}-1}, y_3), \dots, (x_2, y_{N_{\max y}-1}), \dots, (x_{N_{\max x}-1}, y_{N_{\max y}-1})$, is compared with function values of its corresponding two previous adjacent grid points in the X-direction, $z_{(a-1)b}$, and in the Y-direction, $z_{a(b-1)}$, in sequence.

Their x- and y-coordinates are also stored as needed. It is noted that some of these x and y values may be stored more than once. Therefore, all redundant numbers should be removed before or after storing them in two arrays. The method of comparison between the sum of differences of function values and the Z tolerance, Z_t , can be stated as follows:

```

for (a = 2; a < Nmaxx; a++)
{
    for (b = 2; b < Nmaxy; b++)
    {
        zdiff = absolute value of (zab - z(a-1)b) + absolute value of (zab - za(b-1));
        if (zdiff > Zt)
        {
            if a-1 is not in the array XA, store a-1 in XA;
            if a is not in the array XA, store a in XA;
            if b-1 is not in the array YA, store b-1 in YA;
            if b is not in the array YA, store b in YA;
        }
    }
}

```

After comparison and data storage, sorting elements of two arrays XA and YA in incremental order is the next step. Many sorting algorithms have been developed [Knut73, Stin85]. Once the sorting is done, x values in XA become constant values of x for final slicing planes parallel to the Y-Z plane and y values in YA are used for constant values of y for final cutting planes parallel to the X-Z plane. As we can notice, the spacing between any two adjacent parallel planes is most likely not uniform. Therefore, as mentioned before, an algebraic surface can be approximated by an appropriate grid surface with "reasonable" non-uniform intervals between adjacent grid points.

Similarly, a database of non-uniform grid surface representation of an algebraic surface can be directly given for consideration instead of generating data through a sequence of computation and comparison. The format of the database is exactly the same as that for uniform grid surface representation described in Table 2-1.

Published Algorithms

One of the most frequently used methods for removing hidden lines from three-dimensional representations of surface functions of the form $f(x,y,z)=0$, i.e. algebraic surfaces, is the Floating Horizon Algorithm (abbreviated the FHA) [Will72, Wrig73, Roge85]. It converts the 3-D problem to 2-D by intersecting the surface with a series of parallel cutting planes at constant values of x , y or z . In other words, the method can be used to draw only X- or Y- or Z-direction lines in the grid exactly, but not two sets of orthogonal curves.

The algorithm proposed by Anderson [Ande82] does not have the restrictions of the FHA. In his algorithm, an algebraic surface is approximated by a grid surface with a set of grid elements. The algorithm utilizes geometric properties of grid surfaces and their projections to classify three viewing cases on a grid surface depending on the relative viewpoint and to enumerate the facets, i.e. projected images of grid elements. It then computes the vanishing point, pseudoangles and pseudoradii for grid points, initializes a perimeter consisting of always visible facet edges and processes visibilities of other facet edges to modify the perimeter. A table and a figure of execution time data show that this algorithm is exact, linear-time, and fairly simple. However, when the observer's viewpoint lies in a grid plane or lies above a grid point, the algorithm has to treat it as a special case and becomes considerably more complicated.

A more general solution to the imaging problem for quadric surfaces is examined by Blinn [Blin82]. His approach describes in detail its application to a class of surfaces which are closely allied to quadrics but have a wider range of shapes, for example, a surface model of electron density maps of molecular structures. The algorithm is primarily geometric, i.e. it uses standard transformation techniques for quadrics to transform objects into the Viewing Coordinate System and then determines the relative priority between parts of an object or objects so that only visible surface elements will be drawn. Once a visible

surface patch has been found, it is rendered into a frame buffer or on the display device directly.

Based on a ray casting algorithm, an extended method presented by Hanrahan [Hanr83] works in more than three dimensions to produce pictures of algebraic surfaces of polynomial functions. The method uses a symbolic algebra system to automatically derive the equation of intersection between a ray and a general algebraic surface. It then solves this equation using an exact polynomial root-finding algorithm to get the first positive real root of the polynomial within some specified tolerance. The root represents the visible surface element and thus the hidden parts of the surface are eliminated. The methodology is, in fact, very general, placing no restrictions on the number of dimensions or the degree of the surface. However, its calculation of intersections is computationally very expensive.

Motivated by the linearity of computational complexity and problems of Anderson's algorithm [Ande82], a different approach, the "Floating Perimeter Algorithm" [Wang85a] has been developed. The algorithm can exactly and efficiently perform the task of visibility determination on solid objects with partially ordered surface data. For instance, the algorithm can generate images of grid surfaces for algebraic surfaces with hidden-line and hidden-surface elimination no matter where the viewpoint lies. Therefore, Anderson's special cases need no special handling and are as simple as others. Details of the Floating Perimeter Algorithm and the complexity of the algorithm in comparison with that of Anderson's method are discussed in the next section.

An algorithm presented by Roulier [Roul87] is to refine bivariate grid data that is convex and monotonic along the grid lines so that the refined data exhibits the same convexity and monotonicity along the appropriate grid lines. Furthermore, the refined data is consistent with a smooth convex function and can be generated locally. The algorithm is based on some geometric observations about univariate data and an algorithm for shape-preserving quadratic splines for such data [McAl81]. It produces data refinement that can be taken as points on a convex surface or can be used to produce finer data for other

surface-fitting methods. Therefore, the algorithm can be used as is or with standard surface-patch techniques.

Some techniques for performing free-form modeling with algebraic surfaces had been presented by Sederberg [Sede88]. Cubic algebraic surfaces, which are defined by implicit equations of degree three, were specifically emphasized. The classical result that a cubic surface can be defined as the intersection locus of three two-parameter families of planes was explained, and the problem of how to impose derivative continuity between two adjacent algebraic surfaces was also considered.

The Floating Perimeter Algorithm

The Floating Perimeter Algorithm was developed for solving the hidden-line and hidden-surface problems on display of solid objects defined by partially ordered surface data, such as algebraic surfaces of bivariate functions. In the algorithm, a special class of objects, grid surfaces, is used to represent algebraic surfaces. In other words, algebraic surfaces are rendered as graphs represented by their function values on a set of grid points. Grid surfaces and their projections have geometric properties which permit the elimination of hidden lines or hidden surfaces to be done more easily than in the general case.

The algorithm is implemented in the image space and takes advantage of the raster display resolution. The procedure examines the edges of a projected grid surface in a systematic order to find all points visible to the observer which in turn establish a visible periphery for the surface. This changing periphery floats out toward the edges of the visible parts of the solid surface and determines the visibility of every point of the edges yet to be processed. Hence the procedure is termed the Floating Perimeter Algorithm, or, briefly, the FPA.

The FPA consists of the following steps:

(I) Mapping the surface to the Viewing Coordinate System. For a given viewpoint VP (VP_x, VP_y, VP_z), with viewing direction VA (VA_x, VA_y, VA_z), the object surface is viewed

and projected through VP onto a view plane normal to VA. In other words, from the World Coordinate System in which the surface is originally defined and through a sequence of geometric transformations, all grid points of this surface are transformed to the Viewing Coordinate System and then are converted to the Screen Coordinate System. Techniques of the geometric transformations can be found in [Newm79, Fole82, Hear86], and particularly, in [Wang85a]. The projected image size is one of the major factors affecting the complexity of the FPA.

(II) The classification of viewing cases. According to the x- and y-domains of a grid surface, $[X_{min}, X_{max}]$ and $[Y_{min}, Y_{max}]$, the X-Y plane of the world space is partitioned by four planes into nine regions shown in Figure 2-1. The grid surface is defined in the first region. When the viewpoint lies directly above or below this region, the surface is viewed face on. It is viewed edge on if the viewpoint lies in the region 2, 3, 4, or 5 and is viewed corner on from the region 6, 7, 8, or 9. Therefore, viewing an object surface can be classified as one of nine cases depending on the region in which the viewpoint lies.

(III) Enumeration of facets. The enumeration of facets is the key to properly eliminate hidden lines and hidden surfaces for displaying an object surface. Since the grid surface contains no mutually intersecting parts and is defined in 3-D space with Cartesian Coordinates, its facets are enumerated according to the distances between the corresponding grid elements of facets and the projection of the viewpoint on the X-Y plane. Therefore, in this enumeration the z-depth is ignored, only 2-D distance from the X & Y coordinates of each facet to those of the viewpoint, (VP_x, VP_y) , is considered. The nearest facet to the viewpoint is given the smallest number, 1, and should be processed first, while the furthestmost one is given the largest number, i.e. $(N_x-1)(N_y-1)$, and will be considered last.

The facets are enumerated in different order for each viewing case. The enumeration methods for nine viewing cases on a grid surface containing twenty facets are described in Figure 2-2.

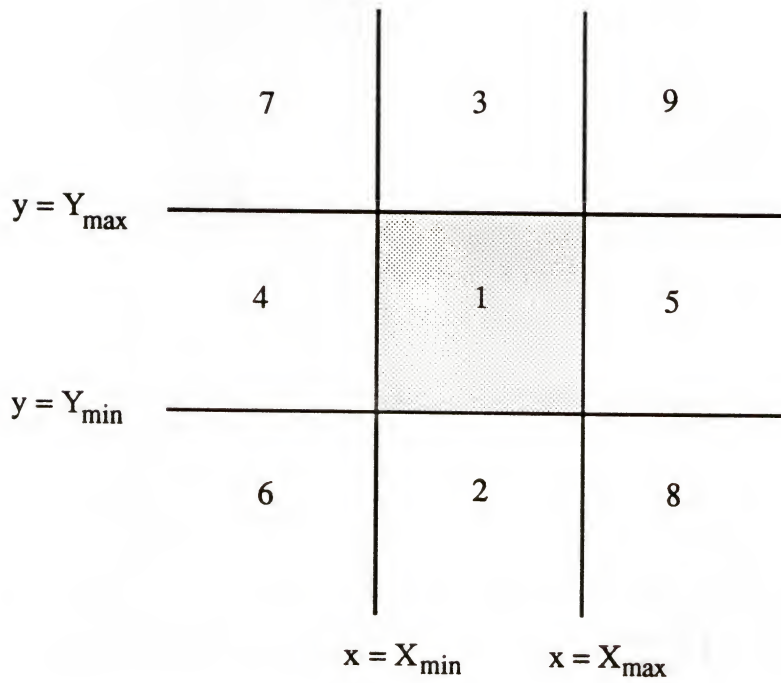


Figure 2-1. 9 regions correspond to 9 viewing cases respectively on the X-Y plane of the world space.

Viewing Case	Enumeration of Facets	Region Classification of Facets																																								
1	<div>$y = Y_{\max}$</div> <table><tr><td>17</td><td>8</td><td>2</td><td>6</td><td>13</td></tr><tr><td>12</td><td>5</td><td>1</td><td>4</td><td>11</td></tr><tr><td>19</td><td>9</td><td>3</td><td>7</td><td>15</td></tr><tr><td>20</td><td>18</td><td>10</td><td>14</td><td>16</td></tr></table> <div>$y = Y_{\min}$</div> <div>$x = X_{\min}$$x = X_{\max}$</div>	17	8	2	6	13	12	5	1	4	11	19	9	3	7	15	20	18	10	14	16	<div>$y = Y_{\max}$</div> <table><tr><td>(8)</td><td>(8)</td><td>(2)</td><td>(6)</td><td>(6)</td></tr><tr><td>(5)</td><td>(5)</td><td>(1)</td><td>(4)</td><td>(4)</td></tr><tr><td>(9)</td><td>(9)</td><td>(3)</td><td>(7)</td><td>(7)</td></tr><tr><td>(9)</td><td>(9)</td><td>(3)</td><td>(7)</td><td>(7)</td></tr></table> <div>$y = Y_{\min}$</div> <div>$x = X_{\min}$$x = X_{\max}$</div>	(8)	(8)	(2)	(6)	(6)	(5)	(5)	(1)	(4)	(4)	(9)	(9)	(3)	(7)	(7)	(9)	(9)	(3)	(7)	(7)
17	8	2	6	13																																						
12	5	1	4	11																																						
19	9	3	7	15																																						
20	18	10	14	16																																						
(8)	(8)	(2)	(6)	(6)																																						
(5)	(5)	(1)	(4)	(4)																																						
(9)	(9)	(3)	(7)	(7)																																						
(9)	(9)	(3)	(7)	(7)																																						
2	<table><tr><td>20</td><td>19</td><td>16</td><td>17</td><td>18</td></tr><tr><td>15</td><td>14</td><td>11</td><td>12</td><td>13</td></tr><tr><td>10</td><td>8</td><td>4</td><td>5</td><td>7</td></tr><tr><td>9</td><td>3</td><td>1</td><td>2</td><td>6</td></tr></table>	20	19	16	17	18	15	14	11	12	13	10	8	4	5	7	9	3	1	2	6	<table><tr><td>(8)</td><td>(8)</td><td>(2)</td><td>(6)</td><td>(6)</td></tr><tr><td>(8)</td><td>(8)</td><td>(2)</td><td>(6)</td><td>(6)</td></tr><tr><td>(8)</td><td>(8)</td><td>(2)</td><td>(6)</td><td>(6)</td></tr><tr><td>(8)</td><td>(8)</td><td>(2)</td><td>(6)</td><td>(6)</td></tr></table>	(8)	(8)	(2)	(6)	(6)	(8)	(8)	(2)	(6)	(6)	(8)	(8)	(2)	(6)	(6)	(8)	(8)	(2)	(6)	(6)
20	19	16	17	18																																						
15	14	11	12	13																																						
10	8	4	5	7																																						
9	3	1	2	6																																						
(8)	(8)	(2)	(6)	(6)																																						
(8)	(8)	(2)	(6)	(6)																																						
(8)	(8)	(2)	(6)	(6)																																						
(8)	(8)	(2)	(6)	(6)																																						
3	<table><tr><td>9</td><td>3</td><td>1</td><td>2</td><td>6</td></tr><tr><td>10</td><td>8</td><td>4</td><td>5</td><td>7</td></tr><tr><td>15</td><td>14</td><td>11</td><td>12</td><td>13</td></tr><tr><td>20</td><td>19</td><td>16</td><td>17</td><td>18</td></tr></table>	9	3	1	2	6	10	8	4	5	7	15	14	11	12	13	20	19	16	17	18	<table><tr><td>(9)</td><td>(9)</td><td>(3)</td><td>(7)</td><td>(7)</td></tr><tr><td>(9)</td><td>(9)</td><td>(3)</td><td>(7)</td><td>(7)</td></tr><tr><td>(9)</td><td>(9)</td><td>(3)</td><td>(7)</td><td>(7)</td></tr><tr><td>(9)</td><td>(9)</td><td>(3)</td><td>(7)</td><td>(7)</td></tr></table>	(9)	(9)	(3)	(7)	(7)	(9)	(9)	(3)	(7)	(7)	(9)	(9)	(3)	(7)	(7)	(9)	(9)	(3)	(7)	(7)
9	3	1	2	6																																						
10	8	4	5	7																																						
15	14	11	12	13																																						
20	19	16	17	18																																						
(9)	(9)	(3)	(7)	(7)																																						
(9)	(9)	(3)	(7)	(7)																																						
(9)	(9)	(3)	(7)	(7)																																						
(9)	(9)	(3)	(7)	(7)																																						
4	<table><tr><td>6</td><td>7</td><td>11</td><td>15</td><td>19</td></tr><tr><td>2</td><td>5</td><td>10</td><td>14</td><td>18</td></tr><tr><td>1</td><td>4</td><td>9</td><td>13</td><td>17</td></tr><tr><td>3</td><td>8</td><td>12</td><td>16</td><td>20</td></tr></table>	6	7	11	15	19	2	5	10	14	18	1	4	9	13	17	3	8	12	16	20	<table><tr><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td></tr><tr><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td></tr><tr><td>(4)</td><td>(4)</td><td>(4)</td><td>(4)</td><td>(4)</td></tr><tr><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td></tr></table>	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(4)	(4)	(4)	(4)	(4)	(7)	(7)	(7)	(7)	(7)
6	7	11	15	19																																						
2	5	10	14	18																																						
1	4	9	13	17																																						
3	8	12	16	20																																						
(6)	(6)	(6)	(6)	(6)																																						
(6)	(6)	(6)	(6)	(6)																																						
(4)	(4)	(4)	(4)	(4)																																						
(7)	(7)	(7)	(7)	(7)																																						
5	<table><tr><td>19</td><td>15</td><td>11</td><td>7</td><td>6</td></tr><tr><td>18</td><td>14</td><td>10</td><td>5</td><td>2</td></tr><tr><td>17</td><td>13</td><td>9</td><td>4</td><td>1</td></tr><tr><td>20</td><td>16</td><td>12</td><td>8</td><td>3</td></tr></table>	19	15	11	7	6	18	14	10	5	2	17	13	9	4	1	20	16	12	8	3	<table><tr><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td></tr><tr><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td></tr><tr><td>(5)</td><td>(5)</td><td>(5)</td><td>(5)</td><td>(5)</td></tr><tr><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td></tr></table>	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(5)	(5)	(5)	(5)	(5)	(9)	(9)	(9)	(9)	(9)
19	15	11	7	6																																						
18	14	10	5	2																																						
17	13	9	4	1																																						
20	16	12	8	3																																						
(8)	(8)	(8)	(8)	(8)																																						
(8)	(8)	(8)	(8)	(8)																																						
(5)	(5)	(5)	(5)	(5)																																						
(9)	(9)	(9)	(9)	(9)																																						

Figure 2-2. The enumeration and region classification of facets in nine viewing cases. The "X" on the grid point (x_i, y_j) indicates the left-bottom grid point of the nearest facet (i, j) to the viewpoint.

Viewing Case	Enumeration of Facets	Region Classification of Facets																																								
6	<div><div>$y = Y_{\max}$</div><table><tr><td>10</td><td>11</td><td>12</td><td>16</td><td>20</td></tr><tr><td>5</td><td>6</td><td>9</td><td>15</td><td>19</td></tr><tr><td>2</td><td>4</td><td>8</td><td>14</td><td>18</td></tr><tr><td>1</td><td>3</td><td>7</td><td>13</td><td>17</td></tr></table><div><div>$y = Y_{\min}$</div><div>$x = X_{\min}$</div><div>$x = X_{\max}$</div></div></div>	10	11	12	16	20	5	6	9	15	19	2	4	8	14	18	1	3	7	13	17	<div><div>$y = Y_{\max}$</div><table><tr><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td></tr><tr><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td></tr><tr><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td></tr><tr><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td><td>(6)</td></tr></table><div><div>$y = Y_{\min}$</div><div>$x = X_{\min}$</div><div>$x = X_{\max}$</div></div></div>	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)	(6)
10	11	12	16	20																																						
5	6	9	15	19																																						
2	4	8	14	18																																						
1	3	7	13	17																																						
(6)	(6)	(6)	(6)	(6)																																						
(6)	(6)	(6)	(6)	(6)																																						
(6)	(6)	(6)	(6)	(6)																																						
(6)	(6)	(6)	(6)	(6)																																						
7	<div><div>$x = X_{\min}$</div><table><tr><td>1</td><td>3</td><td>7</td><td>13</td><td>17</td></tr><tr><td>2</td><td>4</td><td>8</td><td>14</td><td>18</td></tr><tr><td>5</td><td>6</td><td>9</td><td>15</td><td>19</td></tr><tr><td>10</td><td>11</td><td>12</td><td>16</td><td>20</td></tr></table></div>	1	3	7	13	17	2	4	8	14	18	5	6	9	15	19	10	11	12	16	20	<div><div>$x = X_{\min}$</div><table><tr><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td></tr><tr><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td></tr><tr><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td></tr><tr><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td><td>(7)</td></tr></table></div>	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)	(7)
1	3	7	13	17																																						
2	4	8	14	18																																						
5	6	9	15	19																																						
10	11	12	16	20																																						
(7)	(7)	(7)	(7)	(7)																																						
(7)	(7)	(7)	(7)	(7)																																						
(7)	(7)	(7)	(7)	(7)																																						
(7)	(7)	(7)	(7)	(7)																																						
8	<div><table><tr><td>20</td><td>16</td><td>12</td><td>11</td><td>10</td></tr><tr><td>19</td><td>15</td><td>9</td><td>6</td><td>5</td></tr><tr><td>18</td><td>14</td><td>8</td><td>4</td><td>2</td></tr><tr><td>17</td><td>13</td><td>7</td><td>3</td><td>1</td></tr></table><div>$x = X_{\max}$</div></div>	20	16	12	11	10	19	15	9	6	5	18	14	8	4	2	17	13	7	3	1	<div><table><tr><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td></tr><tr><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td></tr><tr><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td></tr><tr><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td><td>(8)</td></tr></table><div>$x = X_{\max}$</div></div>	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)	(8)
20	16	12	11	10																																						
19	15	9	6	5																																						
18	14	8	4	2																																						
17	13	7	3	1																																						
(8)	(8)	(8)	(8)	(8)																																						
(8)	(8)	(8)	(8)	(8)																																						
(8)	(8)	(8)	(8)	(8)																																						
(8)	(8)	(8)	(8)	(8)																																						
9	<div><table><tr><td>17</td><td>13</td><td>7</td><td>3</td><td>1</td></tr><tr><td>18</td><td>14</td><td>8</td><td>4</td><td>2</td></tr><tr><td>19</td><td>15</td><td>9</td><td>6</td><td>5</td></tr><tr><td>20</td><td>16</td><td>12</td><td>11</td><td>10</td></tr></table><div>$x = X_{\max}$</div></div>	17	13	7	3	1	18	14	8	4	2	19	15	9	6	5	20	16	12	11	10	<div><table><tr><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td></tr><tr><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td></tr><tr><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td></tr><tr><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td><td>(9)</td></tr></table><div>$x = X_{\max}$</div></div>	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)	(9)
17	13	7	3	1																																						
18	14	8	4	2																																						
19	15	9	6	5																																						
20	16	12	11	10																																						
(9)	(9)	(9)	(9)	(9)																																						
(9)	(9)	(9)	(9)	(9)																																						
(9)	(9)	(9)	(9)	(9)																																						
(9)	(9)	(9)	(9)	(9)																																						

Figure 2-2--continued.

(IV) The region classification of facets. While enumerating the facets, we also assign a region class number for each of them (see Figure 2-2). Nine region classes, related to the nine viewing cases respectively, are used to classify facets in order to facilitate processing of their edges. The purpose of this region classification for each facet is stated in the next step.

(V) Processing of facets' edges. To display a projected grid surface with hidden lines eliminated in the image space, all edges of facets in the surface image should be considered at least once, and for efficiency just once. Although each facet has four edges, it also shares all its four edges with four adjacent facets. Thus for every facet except the first one, there must exist at least one adjacent facet which had been processed previously, and so at least one of four edges of the current facet can be ignored since it has already been processed. Therefore, while processing facets from the smallest number to the largest, only the first one needs to examine all of its four edges. In the first viewing case, a facet with a region class number 2, 3, 4 or 5 needs to have three edges processed; however, only two edges will be considered for a facet with a region class number 6, 7, 8 or 9. These edges of facets in the first viewing case are shown in Figure 2-3.

(VI) The floating perimeter on visibility determination. In the image space, or the Screen Coordinate System, a floating perimeter is composed of four one-dimensional arrays of boundaries- top, bottom, right and left. The sizes of top and bottom boundary arrays are the same as the horizontal resolution of the raster display, while the sizes of right and left boundary arrays are the same as the device's vertical resolution. Thus on a raster display with a resolution $HR \times VR$ and with the origin (0,0) at its bottom-left corner, these arrays of four boundaries can be initialized as follows:

$$\text{top}[0:HR-1] = 0$$

$$\text{bottom}[0:HR-1] = VR$$

$$\text{right}[0:VR-1] = 0$$

$$\text{left}[0:VR-1] = HR.$$










Region Class	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Edges of Facets to be Processed									

Figure 2-3. The to-be-processed edges of facets in 9 region classes in viewing case 1.

During processing of an edge of a facet, every point on that edge has to be considered. The visibility of a point is determined by its position relative to the floating perimeter. If the point is outside the perimeter, then it is visible, otherwise it is not. A visible flag $vifg[i]$ and an invisible flag $ivfg[i]$ are set for that point to record its visibility. The screen coordinates (x_i, y_j) of visible points are stored in a buffer, $Bbuf[][]$, for updating their corresponding values of the floating perimeter, i.e. four boundaries, later. After all points on the current edge are processed, two other flags, $tvifg$ and $tivfg$, are initialized for that edge and used in logical AND operations with two flags of each point, respectively. These two flag values can tell if the current processing edge is completely visible, partially visible or completely invisible. The pseudo code in Figure 2-4 describes steps in the FPA for the visibility determination of a point and an edge of a facet of a grid surface.

Once all need-to-be-processed edges of a facet have been processed, the total number of visible points, Tvp , on the facet is counted; the x- and y-screen coordinates of these visible points are stored in the boundary buffer $Bbuf[][]$; and all their corresponding values of four boundaries are updated as follows:

```

if (Tvp > 0)
{
    for (i=1; i <= Tvp; i++)
    {
        px = Bbuf[0][i];
        py = Bbuf[1][i];
        top[px]    = MAX (top[px], py);
        bottom[px] = MIN (bottom[px], py);
        right[py]  = MAX (right[py], px);
        left[py]   = MIN (left[py], px);
    }
}.

```

The FPA utilizes the limitation of the image space to a raster display, point-to-point coherence and the object coherence of the connected facets of grid surfaces. Facets are processed in a systematic order, from the nearest one to the furthestmost one, and are used to update the floating perimeter with all visible parts on the facet being processed. This growing floating perimeter then, in turn, determines the visibility of every point of the facet

(a) Visibility Determination on A Point (x_i, y_i) of the Edge Being Processed

```

if ( $x_i \geq \text{right}[y_i] \parallel x_i \leq \text{left}[y_i] \parallel y_i \geq \text{top}[x_i] \parallel y_i \leq \text{bottom}[x_i]$ )
then the point ( $x_i, y_i$ ) is visible, because it is outside the floating perimeter;
    vifg[i] = 1;
    ivfg[i] = 0;
    store the  $x_i, y_i$  in the boundary buffer Bbuf[][];
    if (i == 1) store  $x_i, y_i$  as the beginning of the first visible line segment;
    else
        if (vifg[i-1] == 0) store  $x_i, y_i$  as the start point of a new line segment;
        if (i == edge_length) store  $x_i, y_i$  as the end point of a line segment;
else the point ( $x_i, y_i$ ) is not visible, because it is within the floating perimeter;
    ivfg[i] = 1;
    vifg[i] = 0;
    if (i > 1 && vifg[i-1] == 1)
        the point being processed is invisible, but the previous point is visible;
        thus, store the previous point as the end point of the visible line segment
        being processed.

```

(b) Visibility Determination on An Edge of the Facet Being Processed

```

tvifg = tivfg = 1;
Do logical AND operations with flags vifg[] & ivfg[] of every point on the edge
being processed, respectively:
for (i=1; i <= edge_length; i++)
{
    tvifg = tvifg & vifg[i];
    tivfg = tivfg & ivfg[i];
}
if (tvifg == 1) the edge is completely visible, draw it;
else if (tivfg == 1) the edge is completely invisible;
else the edge is partially visible, draw all visible line segments of this edge.

```

Figure 2-4. The pseudo code in which techniques of the FPA on visibility determination (a) of a point and (b) of an edge of a facet in a grid surface are described.

to be processed next and discards all hidden parts of the facet within the perimeter. Therefore, this algorithm is in fact the inverse of the painter's algorithm for facets processing order and the inverse of window clipping for eliminating invisible parts.

The Floating Perimeter Algorithm is suitable to represent algebraic surfaces as uniform and/or non-uniform grid surfaces with hidden-line and hidden-surface eliminations. Applying the FPA to an algebraic surface of polynomial function $z = -x^4 + x^2 - y^2$, uniform and non-uniform grid surfaces are displayed on a high-communication bandwidth frame buffer [Dres84] as shown in Figures 2-5 and 2-6, respectively. They are both defined over the same rectangle $(-1.2, 1.2 \times -1.2, 1.2)$. The uniform surface (Figure 2-5) consists of $(30-1) \times (30-1)$ uniform straight-edged grid elements; whereas the non-uniform surface is a bit more complicated with the following values: $N_{maxx} = 66$, $N_{maxy} = 42$, $R_x = 3$, $R_y = 3$, and $Z_t = 0.23$. After using techniques of comparison, storage and sorting as mentioned before, we finally obtain $(28-1) \times (28-1)$ non-uniform straight-edged grid elements for the surface of Figure 2-6.

Uniform grid surfaces are more common for representing solid objects with partially ordered surface data, while using non-uniform grid surfaces for visualization is seldom used. However, they can use fewer or the same number of lines to represent the same object in the same or more detail, and, therefore, they require less or the same amount of computation and display than those that are drawn with uniform grid surfaces.

Comparisons of Algorithms

The FPA was implemented in a C program on a VAX-11/780 minicomputer. The program has been tested on many algebraic surfaces. Performance results of the FPA and of Anderson's algorithm are given in Table 2-2 and Table 2-3 for an algebraic surface $f(x,y) = \exp(-x*x - y*y)$ and for a surface consisting of samples from the uniform random distribution on $[0,1]$, respectively. These were generated on uniform grids of varying fineness on the square $[-2,2] \times [-2,2]$ (applies to both the FPA and Anderson's algorithm)

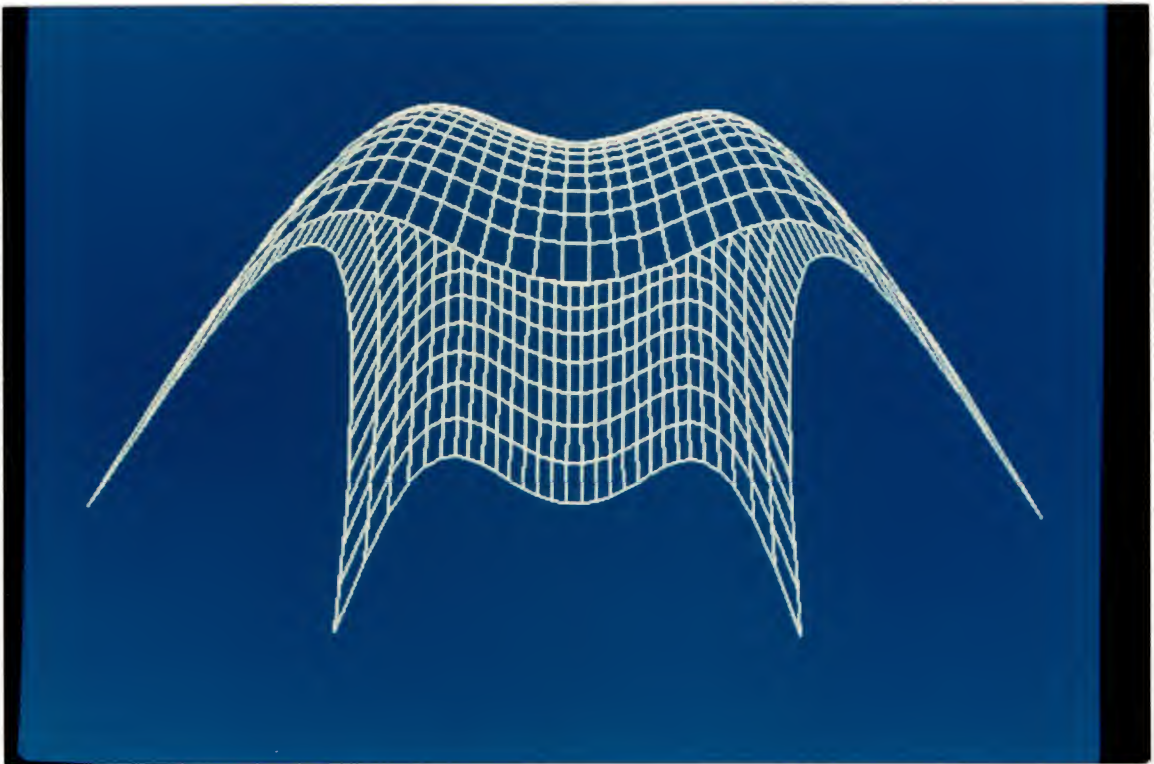


Figure 2-5. An algebraic surface of polynomial function represented as an uniform grid surface.

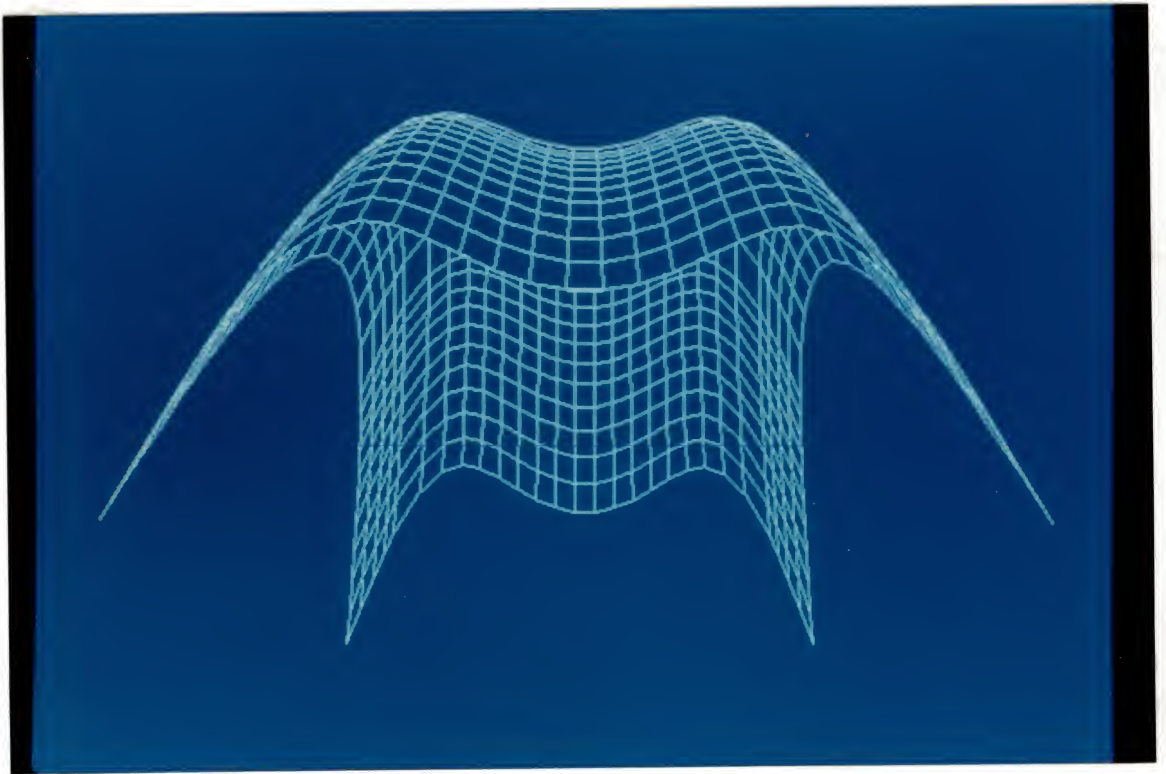


Figure 2-6. An algebraic surface of polynomial function represented as a non-uniform grid surface.

Table 2-2. Execution time (sec) of Anderson's algorithm and of the FPA for the smooth surface.

Grid size	Number of Facets	Number of Edges	<i>the FPA</i> Image Size (number of pixels)						<i>Anderson's Algorithm</i>
			100 ²	200 ²	300 ²	400 ²	500 ²	600 ²	
10 X 10	81	180	0.17	0.30	0.40	0.52	0.58	0.73	0.184
20 X 20	361	760	0.50	0.70	0.87	1.12	1.37	1.57	0.725
30 X 30	841	1740	0.88	1.25	1.57	1.88	2.25	2.58	1.623
40 X 40	1541	3120	1.50	1.90	2.40	2.76	3.23	3.68	2.886
50 X 50	2401	4900	2.20	2.72	3.30	3.80	4.33	4.95	4.527
60 X 60	3481	7080	2.93	3.65	4.30	4.88	5.57	6.18	6.532

Table 2-3. Execution time (sec) of Anderson's algorithm and of the FPA for the random surface.

Grid size	Number of Facets	Number of Edges	<i>the FPA</i>		Image Size (number of pixels)					<i>Anderson's Algorithm</i>
			100 ²	200 ²	300 ²	400 ²	500 ²	600 ²		
10 X 10	81	180	0.22	0.35	0.45	0.63	0.78	0.93	0.221	
20 X 20	361	760	0.62	0.97	1.32	1.67	1.98	2.40	0.921	
30 X 30	841	1740	1.27	1.97	2.62	3.28	3.98	4.67	2.315	
40 X 40	1541	3120	2.20	3.28	4.35	5.45	6.53	7.65	3.969	
50 X 50	2401	4900	3.42	5.13	6.87	8.55	10.22	11.98	6.515	
60 X 60	3481	7080	4.78	7.10	9.48	11.87	14.13	16.28	9.534	

and varying 2-D image size in the range $[100,600]^2$ (applies only to the FPA). The resulting surfaces were drawn from the viewpoint (6,8,3) (see Figure 2-7 & Figure 2-8). From graphs Figure 2-9 and Figure 2-10, it can be seen that the execution time of the FPA, in these cases, appears to increase less than linearly as a function of the number of facets and as a function of the image size, whereas the execution time of Anderson's method is a linear function of the number of facets. To achieve better speed for generating grid surfaces, one might want to implement the FPA approach for a reasonably small number of facets and a small size of the surface image which is then zoomed in large enough for visual inspection. However, if the precision of the image is the issue, one would work with the full image even though it is much slower to do so.

Comparing the performance results of the FPA with that of Anderson's method [Ande82] for the same functions, we found that the FPA is a bit faster. That is because the FPA requires fewer computations, such as comparisons of numbers, calculations of slopes of edges, flag assignments and logical AND operations. The FPA achieves speed at no expense of exactness and generality. On the other hand, Anderson's method needs special handling on some cases which make it more complicated. Furthermore, floating point calculations of vanishing point, pseudoangles and pseudoradii for each grid point of the surface required by his algorithm also make it logically much more complex.

In the FHA, two horizons, upper and lower horizons, are used to set values of two flags to be -1, 0 or 1 for indicating visibilities and positions of the previous and current processing points. Thus nine combinations that result from three values for each of these two flags have to be dealt with separately. Whenever values of two flags are different, the intersection point of the straight line, connecting the previous and current processing points, and one of two horizons must be calculated.

The FPA uses flags `vifg[]` & `ivfg[]` to store values 0 or 1 for each point on the edge being processed to indicate the visibility of the point. If the visibilities of two adjacent points on the same edge are different, the beginning or the end of a visible line segment is

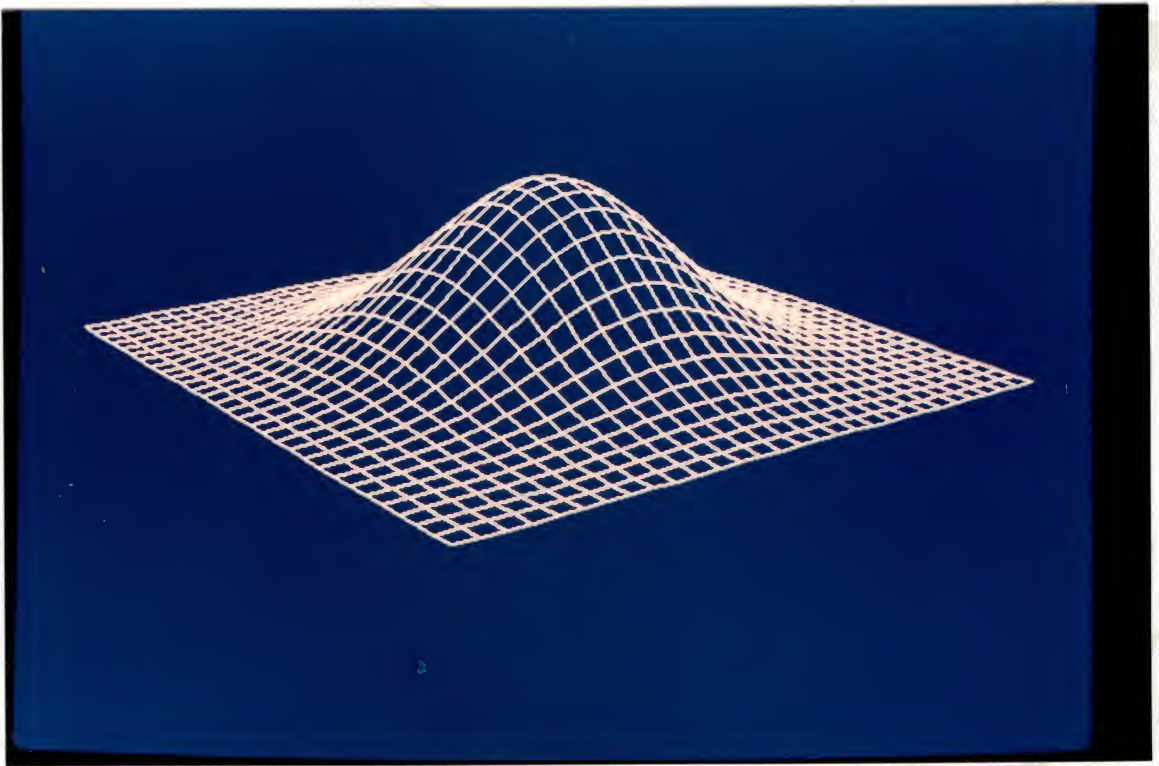


Figure 2-7. The smooth surface used as a test case.

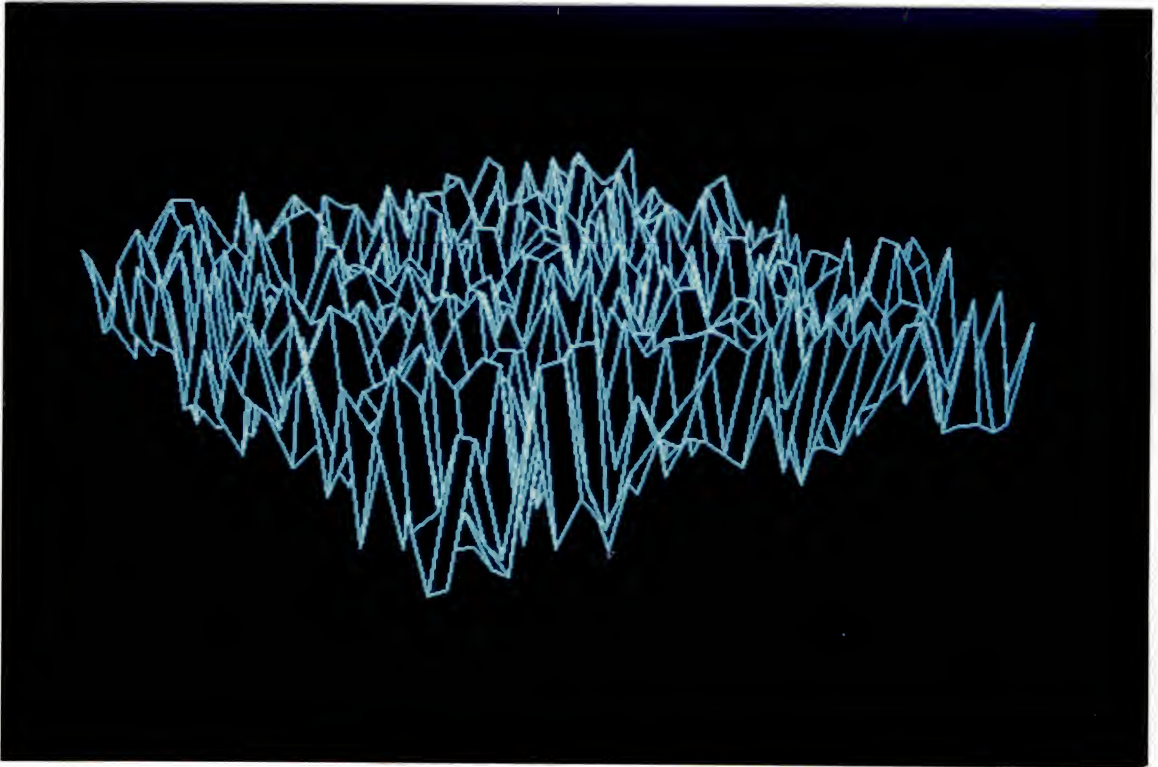


Figure 2-8. The random surface used as a test case.

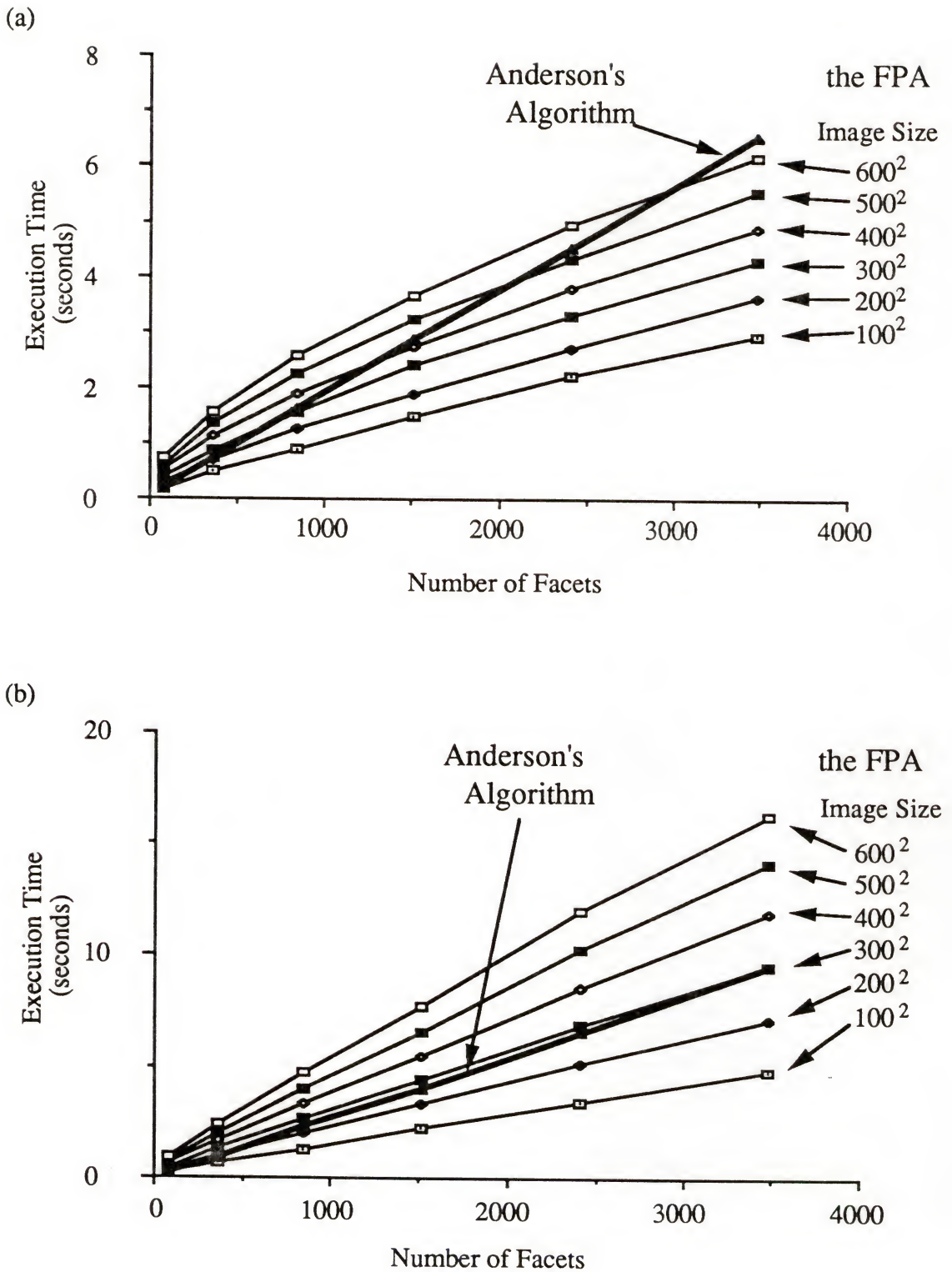
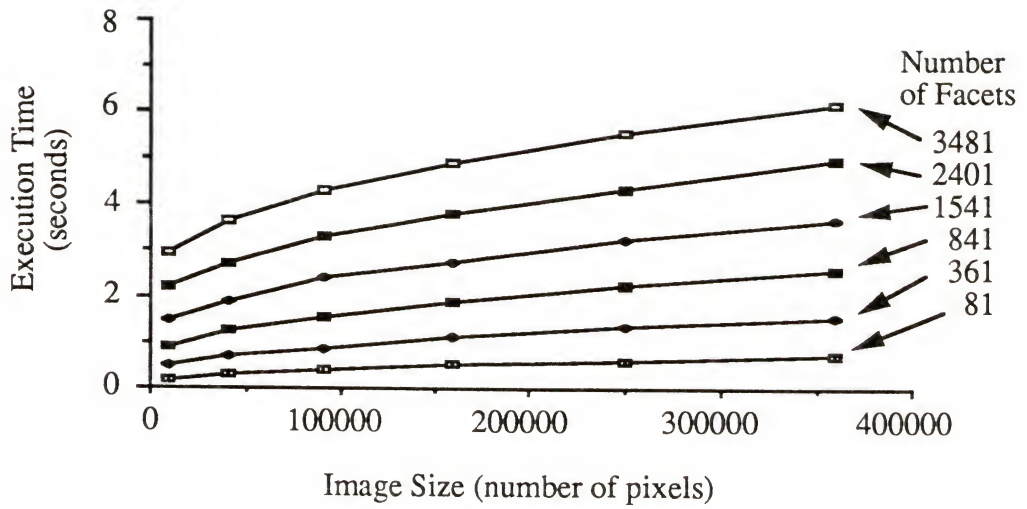


Figure 2-9. Execution time of Anderson's algorithm and of the FPA as functions of the number of facets for (a) a smooth surface and (b) a random surface.

(a)



(b)

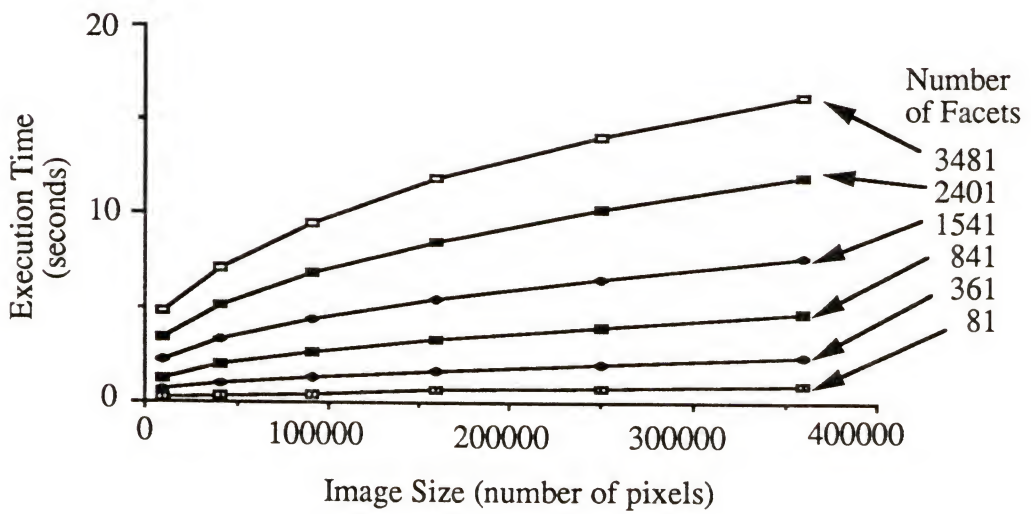


Figure 2-10. Execution time of the FPA as functions of the image size for (a) a smooth surface and (b) a random surface.

stored. Therefore, there is no need for the calculation of the intersection point; this makes the FPA more efficient and simple. An algebraic surface of a bivariate function $f(x,y) = 0.2 \cdot \sin(x) \cdot \cos(y) - 1.5 \cdot \cos(1.75a) \cdot \exp(-a)$, where $a = (x - \pi)^2 + (y - \pi)^2$, which was represented as a set of horizontal curves by means of FHA in [Roge85], is displayed as a grid surface by using the FPA in Figure 2-11. Also Figure 2-12 shows a non-uniform grid surface of this bivariate function.

The storage requirements for the FHA, Anderson's algorithm and the FPA are tabulated in Table 2-4 for comparison. It is obvious that the FHA requires the least space for storing its various parameters, however, it is rather difficult to compare the storage space needed by the other two algorithms. For example, suppose values of parameters are: $N_x=N_y=40$, $HR = 768$ and $VR = 480$, then Anderson's algorithm needs 33,306 bytes for storage, whereas the FPA needs 34,250 bytes. In the case that $N_x = N_y = 50$, Anderson's algorithm needs 51,626 bytes, but the FPA needs only 50,330 bytes for storage. Therefore, these two algorithms require almost the same amount of storage space.

Another comparison for these three algorithms is the ease of changing the viewpoint. Since the FHA only intersects an algebraic surface with a series of parallel cutting planes at constant values of x , y or z and thus draws only X -, Y - or Z -direction lines in the grid exactly, one should be careful about changing the observer's viewpoint in order not to distort the image on the raster display. As for Anderson's algorithm, when the observer's viewpoint lies in a grid plane or lies above a grid point, it has to treat it as a special case and computations becomes considerably more complicated. However, the FPA does not have the restrictions of the other two algorithms. No matter where the viewpoint lies, as long as it is not inside the object, it is always easy for the FPA to determine the visibility of the object defined by partially ordered surface data.

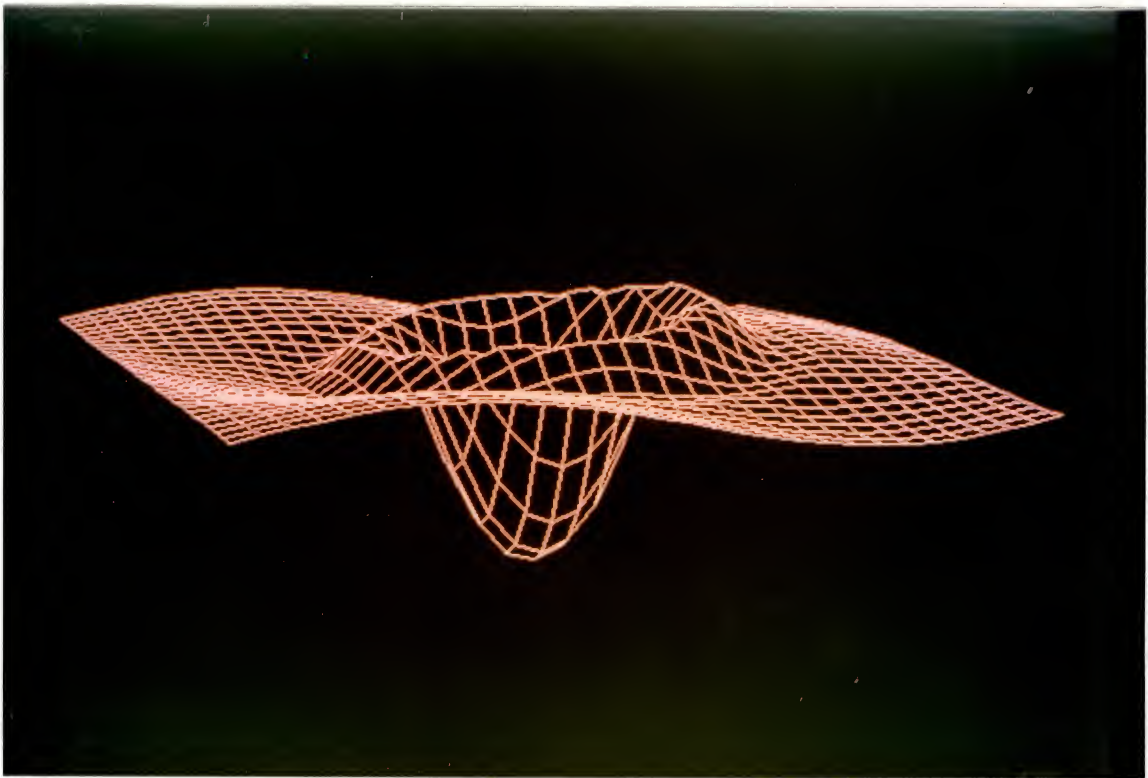


Figure 2-11. Use of the FPA to produce a grid surface of a bivariate function represented by a set of stepped curves by means of the Floating Horizon Algorithm.

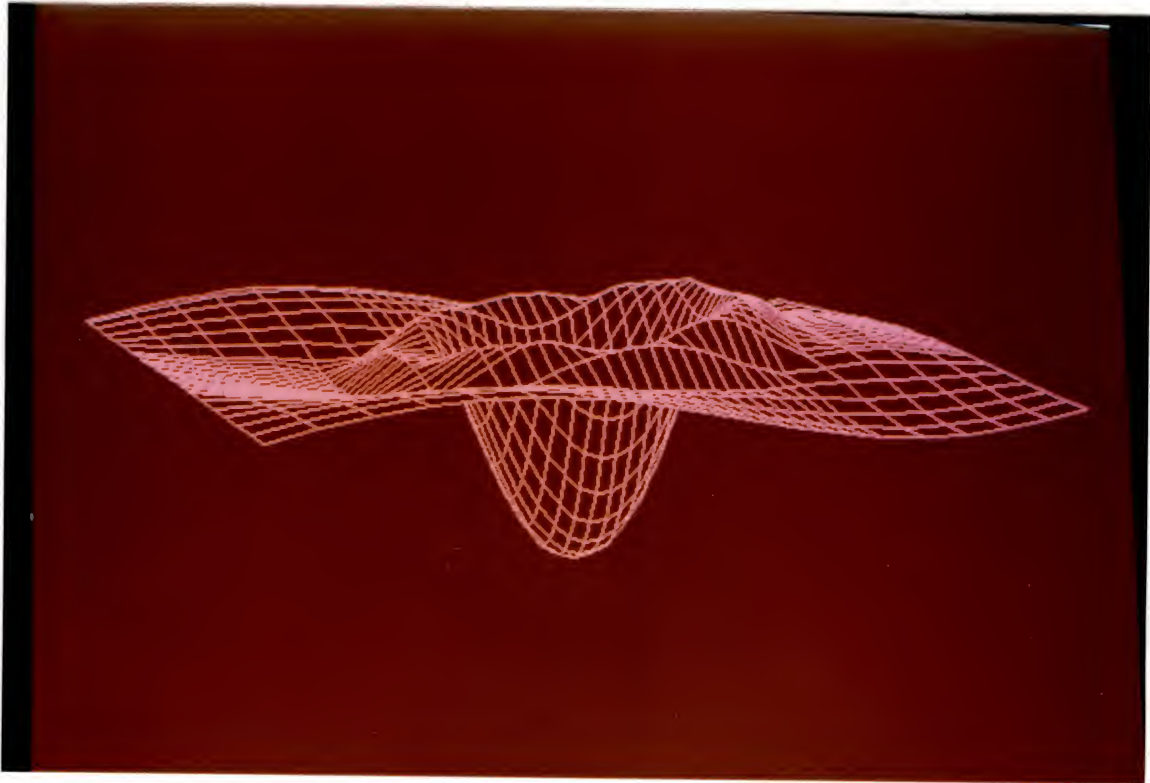


Figure 2-12. A non-uniform grid surface of the same bivariate function as in Figure 2-11.

Table 2-4. The storage requirements for the FHA, Anderson's Algorithm and the FPA. N_x and N_y indicate the number of planes for cutting the surface and parallel to the Y-Z and X-Z planes, respectively; HR and VR are the horizontal and vertical resolutions of the raster display device; N/A means non-applicable.

Storage Purpose	the FHA	Anderson's Algorithm	the FPA
(x, y, z) coordinates of the grid points	float gp[N_x][3];	float gp[N_x][N_y][3];	float gp[N_x][N_y][3];
The viewpoint	float VP[3];	float VP[3];	float VP[3];
The viewing direction	float VA[3];	float VA[3];	float VA[3];
The viewing case	N/A	short view_case;	short view_case;
The image size	N/A	N/A	short image_size;
The horizons or the perimeters	short up[HR]; short low[HR];	short in[(N_x+N_y)*2][2]; short out[(N_x+N_y)*2][2];	short top[HR]; short bottom[HR]; short right[VR]; short left[VR];
Pseudoangles	N/A	float a[N_x][N_y];	N/A
Pseudoradii	N/A	float r[N_x][N_y];	N/A
Facets in the enumerating order with a region class	N/A	N/A	struct facet { short grid[2]; short reg_class; } fcs[(N_x-1)*(N_y-1)];
The visibility flag & the invisibility flag for every point	int Cflag; int Pflag;	N/A N/A	unsigned char vifg[50]; unsigned char ivfg[50];
The visibility flag & the invisibility flag for every edge	N/A N/A	N/A N/A	unsigned char tvifg; unsigned char tivfg;
The total number of visible points and the boundary buffer for processing a facet	N/A N/A	N/A N/A	short Tvp; short Bbuf[200][2];

CHAPTER 3 SHADING

Introduction

The realism of a raster-scan image of a three-dimensional scene depends on the successful simulation of shaded objects. Once visible surfaces have been identified by a hidden-surface algorithm, a shading model is used to calculate the appropriate color and intensity for each point of the surfaces. The shading model takes into account the composition, direction, and geometry of the light sources, the surface orientation and the surface properties of the object.

The light reflected from an object is characterized by being either diffusely or specularly reflected. Diffusely reflected light can be considered as light that has been penetrated below the surface of an object, been absorbed, and then remitted. Diffusely reflected light is scattered equally in all directions. Hence, the position of the observer is unimportant. For instance, dull matte surfaces exhibit diffuse reflection so that the surfaces appear to have the same brightness from all viewing angles.

Specularly reflected light is reflected from the surface of the object. On any shiny surface, such as an apple, a highlight caused by specular reflection is observed. It is also noted that shiny surfaces reflect light unequally in different directions as the highlight moves with the viewing direction. On a perfectly shiny surface, such as a perfect mirror, light is reflected only in the direction for which the angles of incidence and reflection are equal.

In a real scene objects also receive light scattered back to them from the surroundings, e.g., the walls of a room. This ambient light represents a distributed light source which is usually treated as a constant diffuse term by computer graphics illumination models.

To display a visible surface image of objects defined by polygon meshes with any desired shading model described as above, three basic shading techniques- constant shading, intensity interpolation shading, and normal-vector interpolation shading- are commonly adopted. Constant shading uses a single constant normal for each polygon face for calculation of a single intensity value for shading an entire polygon. This shading method is the simplest one, however, it results in intensity discontinuities and faceted appearance of the image. The Mach band effect also accentuates the intensity changes between adjacent facets.

Intensity interpolation shading, usually known from the name of its developer as Gouraud shading [Gour71], eliminates intensity discontinuities and shades the objects with a smoother appearance. The Gouraud shading first determines the intensity at each polygonal vertex with a desired shading model. Then, each polygon is shaded by a linear interpolation of vertex intensities along each edge and finally between edges along each scan line. This shading technique also has deficiencies. The linear shading interpolation of rapid change intensities still induces some Mach band effects. Highlights generated with this technique are anomalous because the shape of the highlight is strongly influenced by the shape of the polygons used to approximate the surface rather than the true surface orientation. If smooth shading is used in a motion sequence, the shading appears to change in strange ways. This occurs because the interpolation basis is fixed to the surface of the screen rather than the surface of the moving objects.

Normal-vector interpolation shading, also called Phong shading [BuiT75], interpolates the surface normal vectors rather than intensities and applies the shading model at each pixel displayed. Although Phong's method requires more computation, it remedies many of the problems with Gouraud shading: greatly reduces the disturbing Mach bands, produces more realistic highlights and largely removes frame-to-frame discontinuities.

A fast Phong shading technique was recently presented by Bishop and Weimer [Bish86]. It uses two dimensional form of the Taylor series to approximate the reflection

equation and then combines this with the forward differences technique. For handling Phong's diffuse-reflection, this method can now evaluate the intensity per pixel with only 2 additions instead of 7 additions, 6 multiplications, 1 division and 1 square root per pixel required by interpolation. Furthermore, Phong's reflection equation, incorporating ambient, diffuse and specular reflectances, can now be evaluated with only 5 additions and 1 memory access per pixel.

Another method called radiosity, originally developed by engineers studying radiative heat transfer [Spar78, Sieg81], was first applied to computer graphics for simple environments, such as an empty room, [Gora84] and then extended to the general case, such as for considering interreflections among objects [Nish85, Cohe85]. The radiosity of a surface is the total light intensity leaving it: the sum of the reflected light and any emitted light if the surface is self-luminous. In the case that all surfaces are diffusely reflecting, the fraction of the radiosity leaving one surface which lands on another is independent of the direction from which incoming energy hits the first surface. This fraction, called the form factor, depends only on the geometrical relationship between the surfaces.

For shading the solid objects with partially ordered surface data, such as grid surfaces, modified approaches to the existing shading techniques have been developed. Details of the methods and some examples of efficiently calculating shading for smoothed surfaces are presented in this chapter.

A Proposed Shading Method

The shading method proposed here uses Phong's shading model [BuiT75], modifies the technique of fast Phong's shading [Bish86] actually employing the Torrance-Sparrow model of specular reflection [Torr67], and thus proposes a new handling of Phong's specular reflection model.

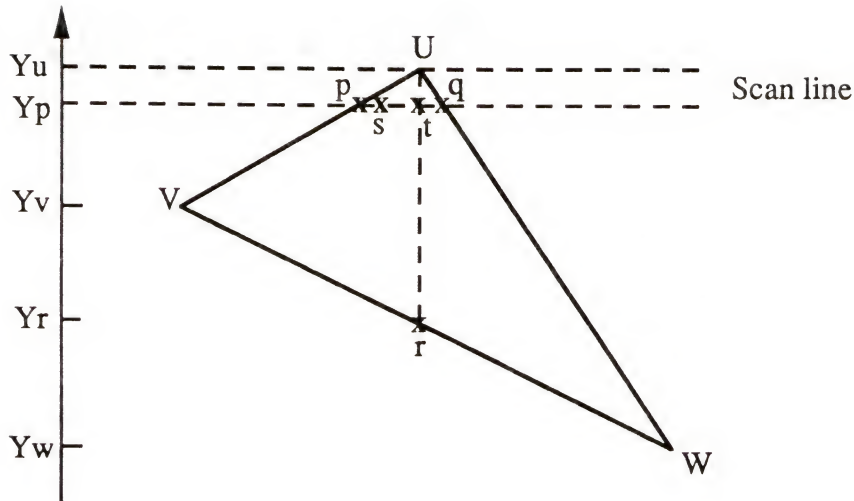
Phong shading, or normal-vector interpolation shading, interpolates the surface normal vectors and applies the shading model at each pixel displayed. A grid surface is composed

of a set of facets, i.e. image of grid elements under a function. Since the four vertices of a facet might not lie on a plane, we split every facet into two triangles. The grid surface is then made up of triangular surface elements. All surface normals at the vertices of each triangle can be directly calculated from the function of the grid surface, which in turn are used to linearly interpolate the approximate surface normal at each pixel across the triangle. This normal-vector interpolation technique is described in Figure 3-1, in which the surface normal at each pixel across the triangle can be approximated by calculating $\mathbf{N}(x,y) = \mathbf{A}x + \mathbf{B}y + \mathbf{C}$.

Phong's reflection equation [Fole82] uses three components--ambient light, diffuse reflection and specular reflection-- to calculate the intensity of light that we should see when we view a surface. These intensity calculations are based on the optical properties of surfaces, the relative positions of the surfaces, and their orientation with respect to the light sources.

$$\begin{aligned}
 I &= I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} \\
 &= I_a * k_a + \frac{I_p}{r + r_0} * (k_d * \cos \theta + k_s * \cos^n \phi) \\
 &= I_a * k_a + \frac{I_p}{r + r_0} * (k_d * (\mathbf{L} \cdot \mathbf{N}) + k_s * (\mathbf{R} \cdot \mathbf{V})^n)
 \end{aligned}$$

where I_a is to account for ambient light; k_a indicates how much of the ambient light is reflected from the object's surfaces; I_p is the intensity of the light source; r is the distance from the perspective viewpoint to the surface; r_0 is a constant that is included to prevent the denominator from approaching zero when r is small; k_d is the coefficient of diffuse reflection; k_s is a constant value due to specular reflection from various kinds of object surface materials; n is a value greater than 1 and depends on surface properties; θ is the angle of incidence between a unit surface normal vector \mathbf{N} and the direction to the light source with a unit vector \mathbf{L} ; ϕ is the viewing angle between the direction for specular reflection with a unit vector \mathbf{R} and the direction of the viewer with a unit vector \mathbf{V} - $\cos \theta$



N_u , N_v , and N_w are the surface normals at the vertices of the polygon UVW. p, q, r, s, and t are 5 points on the polygon. Their X- and Y- coordinates are

$$X_p = X_u - \frac{X_u - X_v}{Y_u - Y_v},$$

$$X_q = X_u - \frac{X_u - X_w}{Y_u - Y_w},$$

$$X_r = X_t = X_u,$$

$$X_s = X_p + 1,$$

$$Y_p = Y_q = Y_s = Y_t = Y_u - 1,$$

$$Y_r = Y_v - (Y_v - Y_w) * \frac{X_u - X_v}{X_w - X_v},$$

and their surface normal vectors can be approximately calculated by interpolation:

$$N_p = N_u * \frac{Y_p - Y_v}{Y_u - Y_v} + N_v * \frac{Y_u - Y_p}{Y_u - Y_v} = N_u + \frac{N_v - N_u}{Y_u - Y_v},$$

$$N_q = N_u + \frac{N_w - N_u}{Y_u - Y_w},$$

$$N_s = N_p + \frac{N_q - N_p}{X_q - X_p},$$

$$N_t = N_u + \frac{N_r - N_u}{Y_u - Y_r},$$

and

$$N_r = N_v * \frac{Y_r - Y_w}{Y_v - Y_w} + N_w * \frac{Y_v - Y_r}{Y_v - Y_w} = N_v + (N_w - N_v) * \frac{Y_v - Y_r}{Y_v - Y_w}.$$

Figure 3-1. Normal-vector interpolation across the polygon from its vertex normals.

Claim:

Suppose the approximate surface normal at each pixel across the polygon can be linearly interpolated from the true surface normals specified at the vertices as

$$\mathbf{N}(x,y) = \mathbf{A}x + \mathbf{B}y + \mathbf{C}$$

where \mathbf{A} , \mathbf{B} , and \mathbf{C} are constant vectors related to the vertex normals.

Solution:

Let the coordinate system be translated (X_u, Y_u) , i.e. make the vertex U be the origin $(0,0)$, then the vectors \mathbf{A} , \mathbf{B} , and \mathbf{C} are calculated as follows:

$$\begin{aligned} \mathbf{A} &= \frac{d\mathbf{N}(x,y)}{dx} = \mathbf{N}_s - \mathbf{N}_p = \frac{\mathbf{N}_q - \mathbf{N}_p}{X_q - X_p} \\ &= \frac{\mathbf{N}_w * (Y_u - Y_v) + \mathbf{N}_u * (Y_v - Y_w) - \mathbf{N}_v * (Y_u - Y_w)}{X_w * (Y_u - Y_v) + X_u * (Y_v - Y_w) - X_v * (Y_u - Y_w)} \end{aligned}$$

$$\begin{aligned} \mathbf{B} &= \frac{d\mathbf{N}(x,y)}{dy} = \mathbf{N}_u - \mathbf{N}_t = \frac{\mathbf{N}_u - \mathbf{N}_r}{Y_u - Y_r} \\ &= \frac{\mathbf{N}_u * (Y_v - Y_w) - \mathbf{N}_v * (Y_r - Y_w) - \mathbf{N}_w * (Y_v - Y_r)}{(Y_u - Y_r) * (Y_v - Y_w)} \end{aligned}$$

$$\mathbf{C} = \mathbf{N}(0,0) = \mathbf{N}_u.$$

Verification:

In the new coordinate system, x - and y -coordinates of points U , V , W and r are

$$\begin{aligned} x_u &= X_u - X_u = 0, & y_u &= Y_u - Y_u = 0, \\ x_v &= X_v - X_u, & y_v &= Y_v - Y_u, \\ x_w &= X_w - X_u, & y_w &= Y_w - Y_u, \\ x_r &= X_r - X_u = 0, & \text{and} & \quad y_r = Y_r - Y_u, \end{aligned}$$

which are then used to replace original coordinates in the vectors \mathbf{A} , \mathbf{B} and \mathbf{C} :

$$\mathbf{A} = \frac{y_v * (\mathbf{N}_u - \mathbf{N}_w) + y_w (\mathbf{N}_v - \mathbf{N}_u)}{x_v * y_w - x_w * y_v}$$

$$\mathbf{B} = \frac{\mathbf{N}_u * (x_w - x_v) + \mathbf{N}_w * x_v - \mathbf{N}_v * x_w}{x_v * y_w - x_w * y_v}$$

$$\mathbf{C} = \mathbf{N}(0,0) = \mathbf{N}_u$$

Therefore,

$$\mathbf{N}(x_u, y_u) = \mathbf{N}(0, 0) = \mathbf{C} = \mathbf{N}_u$$

$$\mathbf{N}(x_v, y_v) = \mathbf{A} * x_v + \mathbf{B} * y_v + \mathbf{C} = \mathbf{N}_v$$

$$\mathbf{N}(x_w, y_w) = \mathbf{A} * x_w + \mathbf{B} * y_w + \mathbf{C} = \mathbf{N}_w.$$

q.e.d.

Figure 3-1--continued.

and $\cos \phi$ can be calculated from the dot products of these unit vectors as shown in Figure 3-2(a). The vector \mathbf{B} in Figure 3-2(b), which is along the bisector of the angle between \mathbf{V} and \mathbf{L} ,

$$\mathbf{B} = \frac{\mathbf{V} + \mathbf{L}}{|\mathbf{V}| + |\mathbf{L}|}$$

is considered in the direction of maximum highlight, i.e. employing Torrance-Sparrow model of specular reflection, in many books and papers, such as [Blin77], [Fole82] and [Bish86], in which vector \mathbf{B} is always used to calculate $\cos \phi$ from the approximation

$$\mathbf{N} \cdot \mathbf{B} \approx \mathbf{R} \cdot \mathbf{V} = \cos \phi.$$

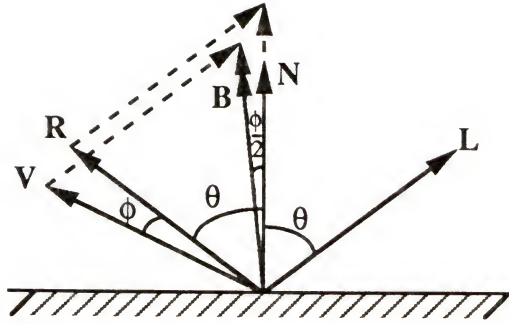
However, the fact is

$$\mathbf{N} \cdot \mathbf{B} = \cos \left(\frac{\phi}{2} \right) \neq \mathbf{R} \cdot \mathbf{V} = \cos \phi.$$

Figure 3-3 shows two more examples to verify it. The comparisons between these two models of specular reflection is shown at the end of this section.

Since the surface normal vector \mathbf{N} at each pixel of a polygon is linearly interpolated from its vertex normals, values of $\cos \theta$ and $\cos \phi$ at each pixel can be calculated more efficiently by combining the interpolation and reflection equations, performing the dot products and expanding the vector magnitude as follows:

$$\begin{aligned} \cos \theta &= \frac{\mathbf{L} \cdot \mathbf{N}}{|\mathbf{L}| \cdot |\mathbf{N}|} = \frac{\mathbf{L} \cdot \mathbf{Ax} + \mathbf{By} + \mathbf{C}}{|\mathbf{L}| \cdot |\mathbf{Ax} + \mathbf{By} + \mathbf{C}|} \\ &= \frac{\mathbf{L} \cdot \mathbf{Ax} + \mathbf{L} \cdot \mathbf{By} + \mathbf{L} \cdot \mathbf{C}}{|\mathbf{L}| * |\mathbf{Ax} + \mathbf{By} + \mathbf{C}|} \\ &= \frac{ax + by + c}{\sqrt{dx^2 + exy + fy^2 + gx + hy + i}} \\ \cos \phi &= \frac{2 * (\mathbf{L} \cdot \mathbf{N}) * (\mathbf{V} \cdot \mathbf{N})}{|\mathbf{L}| * |\mathbf{V}| * |\mathbf{N}| * |\mathbf{N}|} - k_{LV} \\ &= \frac{2 * (\mathbf{L} \cdot (\mathbf{Ax} + \mathbf{By} + \mathbf{C})) * (\mathbf{V} \cdot (\mathbf{Ax} + \mathbf{By} + \mathbf{C}))}{|\mathbf{L}| * |\mathbf{V}| * |\mathbf{Ax} + \mathbf{By} + \mathbf{C}| * |\mathbf{Ax} + \mathbf{By} + \mathbf{C}|} - k_{LV} \\ &= \frac{lx^2 + mxy + oy^2 + px + qy + w}{dx^2 + exy + fy^2 + gx + hy + i} - k_{LV} \end{aligned}$$



(a)

$$\cos \theta = \frac{\mathbf{L}}{|\mathbf{L}|} \cdot \frac{\mathbf{N}}{|\mathbf{N}|}$$

$$\cos \phi = \frac{\mathbf{R}}{|\mathbf{R}|} \cdot \frac{\mathbf{V}}{|\mathbf{V}|}$$

$$\frac{\mathbf{L}}{|\mathbf{L}|} + \frac{\mathbf{R}}{|\mathbf{R}|} = 2 * \cos \theta * \frac{\mathbf{N}}{|\mathbf{N}|} \quad \Rightarrow \quad \frac{\mathbf{R}}{|\mathbf{R}|} = 2 * \cos \theta * \frac{\mathbf{N}}{|\mathbf{N}|} - \frac{\mathbf{L}}{|\mathbf{L}|}$$

$$\begin{aligned} \cos \phi &= \left(2 * \cos \theta * \frac{\mathbf{N}}{|\mathbf{N}|} - \frac{\mathbf{L}}{|\mathbf{L}|} \right) \cdot \frac{\mathbf{V}}{|\mathbf{V}|} = 2 * \cos \theta * \frac{\mathbf{N} \cdot \mathbf{V}}{|\mathbf{N}| * |\mathbf{V}|} - \frac{\mathbf{L} \cdot \mathbf{V}}{|\mathbf{L}| * |\mathbf{V}|} \\ &= 2 * \left(\frac{\mathbf{L}}{|\mathbf{L}|} \cdot \frac{\mathbf{N}}{|\mathbf{N}|} \right) * \left(\frac{\mathbf{N} \cdot \mathbf{V}}{|\mathbf{N}| * |\mathbf{V}|} \right) - \frac{\mathbf{L} \cdot \mathbf{V}}{|\mathbf{L}| * |\mathbf{V}|} \\ &= \frac{2 * (\mathbf{L} \cdot \mathbf{N}) * (\mathbf{V} \cdot \mathbf{N})}{|\mathbf{L}| * |\mathbf{V}| * |\mathbf{N}| * |\mathbf{N}|} - k_{LV} \end{aligned}$$

$$\text{where } k_{LV} = \frac{\mathbf{L} \cdot \mathbf{V}}{|\mathbf{L}| * |\mathbf{V}|} = \text{constant}$$

(b)

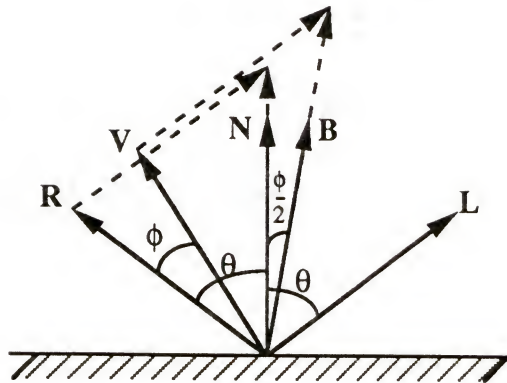
$$\mathbf{B} = \frac{\mathbf{V} + \mathbf{L}}{|\mathbf{V} + \mathbf{L}|}$$

$$\cos \left(\frac{\phi}{2} \right) = \frac{\mathbf{N}}{|\mathbf{N}|} \cdot \frac{\mathbf{B}}{|\mathbf{B}|}$$

$$\begin{aligned} \cos \phi &= 2 * \cos^2 \left(\frac{\phi}{2} \right) - 1 = 2 * \left(\frac{\mathbf{N} \cdot \mathbf{B}}{|\mathbf{N}| * |\mathbf{B}|} \right)^2 - 1 \\ &= \frac{2 * (\mathbf{N} \cdot \mathbf{B}) * (\mathbf{N} \cdot \mathbf{B})}{|\mathbf{N}| * |\mathbf{N}| * |\mathbf{B}| * |\mathbf{B}|} - 1 \end{aligned}$$

Figure 3-2. The angle of incidence, θ , the angle of specular reflection for an ideal reflector, θ , and the viewing angle ϕ .

(a)



(b)

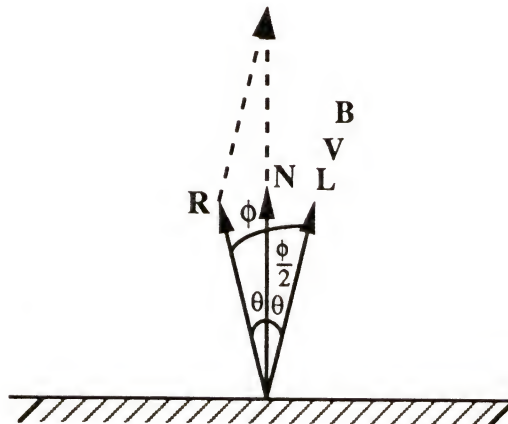


Figure 3-3. Two more examples to show that $\mathbf{N} \cdot \mathbf{B} = \cos\left(\frac{\phi}{2}\right) \neq \cos \phi$.

where

$$A = \frac{N_w * (Y_u - Y_v) + N_u * (Y_v - Y_w) - N_v * (Y_u - Y_w)}{X_w * (Y_u - Y_v) + X_u * (Y_v - Y_w) - X_v * (Y_u - Y_w)}$$

$$B = \frac{N_u * (Y_v - Y_w) - N_v * (Y_r - Y_w) - N_w * (Y_v - Y_r)}{(Y_u - Y_r) * (Y_v - Y_w)}$$

$$C = N(0,0) = N_u$$

$$k_{LV} = \frac{\mathbf{L} \cdot \mathbf{V}}{|\mathbf{L}| * |\mathbf{V}|} = \text{constant}$$

$$a = \frac{\mathbf{L} \cdot \mathbf{A}}{|\mathbf{L}|}$$

$$b = \frac{\mathbf{L} \cdot \mathbf{B}}{|\mathbf{L}|}$$

$$c = \frac{\mathbf{L} \cdot \mathbf{C}}{|\mathbf{L}|}$$

$$d = \mathbf{A} \cdot \mathbf{A}$$

$$e = 2 \mathbf{A} \cdot \mathbf{B}$$

$$f = \mathbf{B} \cdot \mathbf{B}$$

$$g = 2 \mathbf{A} \cdot \mathbf{C}$$

$$h = 2 \mathbf{B} \cdot \mathbf{C}$$

$$i = \mathbf{C} \cdot \mathbf{C}$$

$$aa = \frac{\mathbf{V} \cdot \mathbf{A}}{|\mathbf{V}|}$$

$$bb = \frac{\mathbf{V} \cdot \mathbf{B}}{|\mathbf{V}|}$$

$$cc = \frac{\mathbf{V} \cdot \mathbf{C}}{|\mathbf{V}|}$$

$$l = 2 * a * aa$$

$$m = 2 * (a * bb + b * aa)$$

$$o = 2 * b * bb$$

$$p = 2 * (a * cc + c * aa)$$

$$q = 2 * (b * cc + c * bb)$$

$$w = 2 * c * cc.$$

By using forward differences, $\cos \theta$ can be evaluated for successive values of x and y with 3 additions, 1 division and 1 square root per pixel, while $\cos \phi$ requires 4 additions and 1 division per pixel. These are substantial savings over Phong's formulation, however, they can be even more rapidly evaluated by using a good approximation.

Taylor's series for a function of two variables about the point (s,t) is

$$\begin{aligned} f(x,y) &= f(s+u,t+v) = \sum_{n=0}^{\infty} \frac{1}{n!} \left(u \frac{\partial}{\partial x} + v \frac{\partial}{\partial y} \right)^n f(s,t) \\ &= f(s,t) + \left(u \frac{\partial}{\partial x} + v \frac{\partial}{\partial y} \right) f(s,t) + \frac{1}{2!} \left(u \frac{\partial}{\partial x} + v \frac{\partial}{\partial y} \right)^2 f(s,t) + \dots \end{aligned}$$

Let $s = t = 0$, then $u = x - s = x$, $v = y - t = y$, and

$$\begin{aligned} f(x,y) &= f(0,0) + xf_x(0,0) + yf_y(0,0) + \frac{1}{2} x^2 f_{xx}(0,0) + xyf_{xy}(0,0) \\ &\quad + \frac{1}{2} y^2 f_{yy}(0,0) + \dots \text{ (terms of order greater than two).} \end{aligned}$$

Using Taylor's series to represent the above functions of $\cos \theta$ and $\cos \phi$ and expanding them to the second degree produce

$$\cos \theta = f_1(x,y) = \frac{ax + by + c}{\sqrt{dx^2 + exy + fy^2 + gx + hy + i}}$$

$$\cos \theta \approx T_5 x^2 + T_4 xy + T_3 y^2 + T_2 x + T_1 y + T_0$$

with

$$T_5 = \frac{3cg^2 - 4cdi - 4agi}{8i^2\sqrt{i}}$$

$$T_4 = \frac{3cgh - 2cei - 2bgi - 2ahi}{4i^2\sqrt{i}}$$

$$T_3 = \frac{3ch^2 - 4cfi - 4bhi}{8i^2\sqrt{i}}$$

$$T_2 = \frac{2ai - cg}{2i\sqrt{i}}$$

$$T_1 = \frac{2bi - ch}{2i\sqrt{i}}$$

$$T_0 = \frac{c}{\sqrt{i}}$$

and

$$\cos \phi = f_2(x, y) = \frac{lx^2 + mxy + oy^2 + px + qy + w}{dx^2 + exy + fy^2 + gx + hy + i} - k_{LV}$$

$$\cos \phi \approx T_{55}x^2 + T_{44}xy + T_{33}y^2 + T_{22}x + T_{11}y + T_{00}$$

with

$$T_{55} = \frac{2li^4 - 2dwi^3 - 2gpi^3 - 2g^2w}{i^5}$$

$$T_{44} = \frac{mi^4 - ewi^3 - hpi^3 - gqi^3 - 2ghw}{i^5}$$

$$T_{33} = \frac{2oi^4 - 2fwi^3 - 2hqi^3 - 2h^2w}{i^5}$$

$$T_{22} = \frac{ip - gw}{i^2}$$

$$T_{11} = \frac{iq - hw}{i^2}$$

$$T_{00} = \frac{w}{i} - k_{LV}.$$

Although all the terms of order greater than two in both functions f_1 and f_2 are eliminated, the "errors" introduced by these approximations are of no great consequence since Phong's normal interpolation and the reflection equation are already approximations. Using forward differences, both of $\cos \theta$ and $\cos \phi$ can be evaluated for successive values of x and y with only 2 additions per pixel.

For evaluating the intensity of the light at each pixel in a polygon, which is usually a colored surface, three color components of the intensity should be separately calculated. For an RGB video monitor, color components are red, green and blue. Parameters for intensity and reflectivity then become three-element tuples, with one element for each of the color components. For instance, the triple representing the coefficient of diffuse reflection is (k_{dr}, k_{dg}, k_{db}) . However, since the constant value k_s for specular reflection is not

dependent on the color of the surface, k_s is the same for all three color components. Hence, the intensity calculation breaks into three equations:

$$I_r = I_{ar} * k_{ar} + \frac{I_{pr}}{r + r_0} * (k_{dr} * \cos \theta + k_s * \cos^n \phi)$$

$$I_g = I_{ag} * k_{ag} + \frac{I_{pg}}{r + r_0} * (k_{dg} * \cos \theta + k_s * \cos^n \phi)$$

$$I_b = I_{ab} * k_{ab} + \frac{I_{pb}}{r + r_0} * (k_{db} * \cos \theta + k_s * \cos^n \phi).$$

Diffuse Reflection and Ambient Light

The intensity for ambient light and diffuse reflection can be calculated together. For the red component,

$$I_{r1} = I_{ar} * k_{ar} + \frac{I_{pr}}{r + r_0} * k_{dr} * \cos \theta,$$

the intensity at each pixel of a polygon can be rewritten as

$$I_{r1}(x, y) = I_{ar} * k_{ar} + \frac{I_{pr}}{r + r_0} * k_{dr} * (T_5 x^2 + T_4 xy + T_3 y^2 + T_2 x + T_1 y + T_0).$$

Combining the constant ambient light value with the varying diffuse reflection yields

$$I_{r1}(x, y) = W_5 x^2 + W_4 xy + W_3 y^2 + W_2 x + W_1 y + W_0$$

with

$$W_5 = T_5 * k k_{dr}$$

$$W_4 = T_4 * k k_{dr}$$

$$W_3 = T_3 * k k_{dr}$$

$$W_2 = T_2 * k k_{dr}$$

$$W_1 = T_1 * k k_{dr}$$

$$W_0 = T_0 * k k_{dr} + I_{ar} * k_{ar}$$

where $k k_{dr} = \frac{I_{pr}}{r + r_0} * k_{dr}.$

Using forward differences, $I_{r1}(x, y)$ can be evaluated for successive values of x and y with only 2 additions per pixel. Similarly, the intensity for the green and blue components can also be evaluated for successive values of x and y with 2 additions per pixel. However, in some special cases, such as a green surface with the only nonzero value for the green diffuse reflectivity component, k_{dg} , and two zeros for k_{dr} and k_{db} , we need only to calculate $I_{g1}(x, y)$ for each pixel while the other two intensity components, I_{r1} and I_{b1} , are constants. In other words, only 2 additions are needed in this special case for evaluating the intensity of ambient light and diffuse reflection at each pixel for successive values of x and y .

Specular Reflection

The specular reflection is dependent on the light source. Normally, objects are illuminated with white light, so that the specular reflection is a bright white spot, i.e. the three color components of intensity for specular reflection are the same, $I_{r2} = I_{g2} = I_{b2}$, because the white light source has the property of $I_{pr} = I_{pg} = I_{pb}$. In general, the red component is calculated as follows:

$$I_{r2} = \frac{I_{pr}}{r + r_0} * k_s * \cos^n \phi$$

which can be rewritten as

$$I_{r2} = I_{sr} * \text{pow}(\cos \phi, n)$$

where

$$I_{sr} = \frac{I_{pr} * k_s}{r + r_0} = \text{constant} \quad \text{and}$$

$\text{pow}(\)$ is the power function, i.e. $\text{pow}(\cos \phi, n) = \cos^n \phi$.

Since $0 \leq k_s \leq 1$, $r + r_0 \geq 1$, $n > 1$ and $0 \leq \text{pow}(\cos \phi, n) \leq 1$, it is obvious that $0 \leq I_{r2} \leq I_{pr}$. The constant value I_{sr} can be calculated at the very beginning. Suppose I_{pr} has a common maximum intensity level 255, k_s is equal to 1, and $r + r_0 = 1$, then I_{sr} also has the value of 255 and I_{r2} has a value in the range of (0, 255) depending on the value n .

and $\cos \phi$. As mentioned above, the value of $\cos \phi$ can be evaluated for successive values of x and y with 2 additions per pixel, which can then be used to obtain the value of I_{r2} by the following procedures:

$$\cos \phi_j = \text{pow} \left(\frac{j}{255}, \frac{1}{n} \right), \quad j = 1, 2, \dots, 255$$

$$I_{r2} = \begin{cases} 0 & \text{if } \cos \phi < \cos \phi_1 \\ j & \text{if } \cos \phi_j < \cos \phi < \cos \phi_{j+1} \\ 255 & \text{if } \cos \phi = 1 \end{cases}$$

where the values of $\cos \phi_j$ can be computed just once for any given surface and stored in a look-up table. Table 3-1 and 3-2 are two examples for $n = 6$ and $n = 8$, respectively. Whenever k_s is not equal to 1 or $r + r_0 \neq 1$, the size of the look-up table, j , is changed and all values of $\cos \phi_j$ also vary. This variation can be seen in Table 3-3 and 3-4 corresponding to Table 3-1 and 3-2, respectively. By performing comparisons with values of the power function in a look-up table instead of calculating exponentiation and multiplication, the intensity for specular reflection can be evaluated with significant time savings.

After evaluating the three color components of intensity for ambient light, diffuse reflection and specular reflection separately, the total intensity can be obtained by adding them together, i.e. 1 addition for each of red, green and blue color components. According to the above proposed modified fast Phong shading approach, some grid surfaces with different optical properties and orientation, different direction of the light source and the viewer are shaded in Figure 3-4 and 3-5. With the other model of specular reflection as mentioned above, i.e. using $\mathbf{N} \cdot \mathbf{B} (= \cos \left(\frac{\phi}{2} \right))$ instead of $\mathbf{R} \cdot \mathbf{V} (= \cos \phi)$, these surfaces are shaded in Figure 3-6 and 3-7, respectively, for comparison. For specular reflection, both of the shaded surfaces in Figure 3-4 and 3-6 use the same look-up table as shown in Table 3-4, whereas the surfaces of Figure 3-5 and 3-7 apply Table 3-3.

Table 3-1. A look-up table storing values of $\cos \phi_j$ for specular reflection with $I_p = 255$, $r + r_0 = 1$, $k_s = 1$ and $n = 6$.

j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$
1	0.3971	52	0.7672	103	0.8598	154	0.9194	205	0.9643
2	0.4457	53	0.7696	104	0.8612	155	0.9204	206	0.9651
3	0.4769	54	0.7720	105	0.8625	156	0.9214	207	0.9658
4	0.5003	55	0.7744	106	0.8639	157	0.9223	208	0.9666
5	0.5193	56	0.7767	107	0.8652	158	0.9233	209	0.9674
6	0.5353	57	0.7790	108	0.8666	159	0.9243	210	0.9682
7	0.5492	58	0.7813	109	0.8679	160	0.9253	211	0.9689
8	0.5616	59	0.7835	110	0.8692	161	0.9262	212	0.9697
9	0.5727	60	0.7857	111	0.8706	162	0.9272	213	0.9705
10	0.5829	61	0.7879	112	0.8719	163	0.9281	214	0.9712
11	0.5922	62	0.7900	113	0.8732	164	0.9291	215	0.9720
12	0.6009	63	0.7921	114	0.8744	165	0.9300	216	0.9727
13	0.6089	64	0.7942	115	0.8757	166	0.9310	217	0.9735
14	0.6165	65	0.7963	116	0.8770	167	0.9319	218	0.9742
15	0.6236	66	0.7983	117	0.8782	168	0.9328	219	0.9750
16	0.6304	67	0.8003	118	0.8795	169	0.9337	220	0.9757
17	0.6368	68	0.8023	119	0.8807	170	0.9347	221	0.9764
18	0.6429	69	0.8042	120	0.8819	171	0.9356	222	0.9772
19	0.6487	70	0.8062	121	0.8832	172	0.9365	223	0.9779
20	0.6543	71	0.8081	122	0.8844	173	0.9374	224	0.9786
21	0.6596	72	0.8100	123	0.8856	174	0.9383	225	0.9794
22	0.6647	73	0.8118	124	0.8868	175	0.9392	226	0.9801
23	0.6697	74	0.8137	125	0.8880	176	0.9401	227	0.9808
24	0.6744	75	0.8155	126	0.8891	177	0.9410	228	0.9815
25	0.6790	76	0.8173	127	0.8903	178	0.9418	229	0.9822
26	0.6835	77	0.8191	128	0.8915	179	0.9427	230	0.9829
27	0.6878	78	0.8208	129	0.8926	180	0.9436	231	0.9837
28	0.6920	79	0.8226	130	0.8938	181	0.9445	232	0.9844
29	0.6961	80	0.8243	131	0.8949	182	0.9453	233	0.9851
30	0.7000	81	0.8260	132	0.8961	183	0.9462	234	0.9858
31	0.7038	82	0.8277	133	0.8972	184	0.9471	235	0.9865
32	0.7076	83	0.8294	134	0.8983	185	0.9479	236	0.9872
33	0.7112	84	0.8310	135	0.8994	186	0.9488	237	0.9879
34	0.7148	85	0.8327	136	0.9005	187	0.9496	238	0.9886
35	0.7182	86	0.8343	137	0.9016	188	0.9505	239	0.9893
36	0.7216	87	0.8359	138	0.9027	189	0.9513	240	0.9899
37	0.7249	88	0.8375	139	0.9038	190	0.9521	241	0.9906
38	0.7281	89	0.8391	140	0.9049	191	0.9530	242	0.9913
39	0.7313	90	0.8407	141	0.9060	192	0.9538	243	0.9920
40	0.7344	91	0.8422	142	0.9070	193	0.9546	244	0.9927
41	0.7374	92	0.8437	143	0.9081	194	0.9555	245	0.9934
42	0.7404	93	0.8453	144	0.9092	195	0.9563	246	0.9940
43	0.7433	94	0.8468	145	0.9102	196	0.9571	247	0.9947
44	0.7461	95	0.8483	146	0.9112	197	0.9579	248	0.9954
45	0.7489	96	0.8497	147	0.9123	198	0.9587	249	0.9960
46	0.7517	97	0.8512	148	0.9133	199	0.9595	250	0.9967
47	0.7544	98	0.8527	149	0.9143	200	0.9603	251	0.9974
48	0.7570	99	0.8541	150	0.9154	201	0.9611	252	0.9980
49	0.7596	100	0.8555	151	0.9164	202	0.9619	253	0.9987
50	0.7622	101	0.8570	152	0.9174	203	0.9627	254	0.9993
51	0.7647	102	0.8584	153	0.9184	204	0.9635	255	1.0000

Table 3-2. A look-up table storing values of $\cos \phi_j$ for specular reflection with $I_p = 255$, $r + r_0 = 1$, $k_s = 1$ and $n = 8$.

j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$
1	0.5002	52	0.8198	103	0.8929	154	0.9389	205	0.9731
2	0.5455	53	0.8217	104	0.8939	155	0.9397	206	0.9737
3	0.5739	54	0.8236	105	0.8950	156	0.9404	207	0.9743
4	0.5949	55	0.8255	106	0.8961	157	0.9412	208	0.9749
5	0.6117	56	0.8274	107	0.8971	158	0.9419	209	0.9754
6	0.6258	57	0.8292	108	0.8982	159	0.9427	210	0.9760
7	0.6380	58	0.8310	109	0.8992	160	0.9434	211	0.9766
8	0.6487	59	0.8328	110	0.9002	161	0.9441	212	0.9772
9	0.6584	60	0.8345	111	0.9013	162	0.9449	213	0.9778
10	0.6671	61	0.8363	112	0.9023	163	0.9456	214	0.9783
11	0.6751	62	0.8380	113	0.9033	164	0.9463	215	0.9789
12	0.6825	63	0.8397	114	0.9043	165	0.9470	216	0.9795
13	0.6893	64	0.8413	115	0.9053	166	0.9478	217	0.9800
14	0.6957	65	0.8429	116	0.9062	167	0.9485	218	0.9806
15	0.7018	66	0.8446	117	0.9072	168	0.9492	219	0.9812
16	0.7075	67	0.8461	118	0.9082	169	0.9499	220	0.9817
17	0.7128	68	0.8477	119	0.9091	170	0.9506	221	0.9823
18	0.7179	69	0.8493	120	0.9101	171	0.9513	222	0.9828
19	0.7228	70	0.8508	121	0.9110	172	0.9520	223	0.9834
20	0.7275	71	0.8523	122	0.9120	173	0.9527	224	0.9839
21	0.7319	72	0.8538	123	0.9129	174	0.9533	225	0.9845
22	0.7362	73	0.8553	124	0.9138	175	0.9540	226	0.9850
23	0.7403	74	0.8567	125	0.9147	176	0.9547	227	0.9856
24	0.7442	75	0.8582	126	0.9156	177	0.9554	228	0.9861
25	0.7480	76	0.8596	127	0.9166	178	0.9561	229	0.9866
26	0.7517	77	0.8610	128	0.9175	179	0.9567	230	0.9872
27	0.7553	78	0.8624	129	0.9183	180	0.9574	231	0.9877
28	0.7587	79	0.8637	130	0.9192	181	0.9581	232	0.9883
29	0.7620	80	0.8651	131	0.9201	182	0.9587	233	0.9888
30	0.7653	81	0.8664	132	0.9210	183	0.9594	234	0.9893
31	0.7684	82	0.8678	133	0.9219	184	0.9600	235	0.9898
32	0.7715	83	0.8691	134	0.9227	185	0.9607	236	0.9904
33	0.7745	84	0.8704	135	0.9236	186	0.9613	237	0.9909
34	0.7774	85	0.8717	136	0.9244	187	0.9620	238	0.9914
35	0.7802	86	0.8730	137	0.9253	188	0.9626	239	0.9919
36	0.7829	87	0.8742	138	0.9261	189	0.9633	240	0.9925
37	0.7856	88	0.8755	139	0.9270	190	0.9639	241	0.9930
38	0.7882	89	0.8767	140	0.9278	191	0.9645	242	0.9935
39	0.7908	90	0.8779	141	0.9286	192	0.9652	243	0.9940
40	0.7933	91	0.8791	142	0.9294	193	0.9658	244	0.9945
41	0.7958	92	0.8804	143	0.9302	194	0.9664	245	0.9950
42	0.7982	93	0.8815	144	0.9311	195	0.9670	246	0.9955
43	0.8005	94	0.8827	145	0.9319	196	0.9676	247	0.9960
44	0.8028	95	0.8839	146	0.9327	197	0.9683	248	0.9965
45	0.8051	96	0.8850	147	0.9335	198	0.9689	249	0.9970
46	0.8073	97	0.8862	148	0.9343	199	0.9695	250	0.9975
47	0.8095	98	0.8873	149	0.9350	200	0.9701	251	0.9980
48	0.8116	99	0.8885	150	0.9358	201	0.9707	252	0.9985
49	0.8137	100	0.8896	151	0.9366	202	0.9713	253	0.9990
50	0.8157	101	0.8907	152	0.9374	203	0.9719	254	0.9995
51	0.8178	102	0.8918	153	0.9381	204	0.9725	255	1.0000

Table 3-3. A look-up table storing values of $\cos \phi_j$ for specular reflection with $I_p = 255$, $r + r_0 = 1.2$, $k_s = 0.5$ and $n = 6$.

j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$
1	0.4597	23	0.7752	45	0.8669	67	0.9264	89	0.9713
2	0.5160	24	0.7807	46	0.8701	68	0.9287	90	0.9731
3	0.5520	25	0.7860	47	0.8732	69	0.9309	91	0.9749
4	0.5792	26	0.7912	48	0.8763	70	0.9332	92	0.9767
5	0.6011	27	0.7962	49	0.8793	71	0.9354	93	0.9784
6	0.6196	28	0.8010	50	0.8823	72	0.9376	94	0.9802
7	0.6358	29	0.8057	51	0.8852	73	0.9397	95	0.9819
8	0.6501	30	0.8103	52	0.8881	74	0.9419	96	0.9836
9	0.6630	31	0.8147	53	0.8909	75	0.9440	97	0.9853
10	0.6747	32	0.8190	54	0.8937	76	0.9461	98	0.9870
11	0.6855	33	0.8233	55	0.8964	77	0.9481	99	0.9887
12	0.6955	34	0.8274	56	0.8991	78	0.9502	100	0.9903
13	0.7049	35	0.8314	57	0.9018	79	0.9522	101	0.9920
14	0.7136	36	0.8353	58	0.9044	80	0.9542	102	0.9936
15	0.7219	37	0.8391	59	0.9070	81	0.9562	103	0.9952
16	0.7297	38	0.8428	60	0.9095	82	0.9581	104	0.9968
17	0.7371	39	0.8465	61	0.9120	83	0.9601	105	0.9984
18	0.7442	40	0.8501	62	0.9145	84	0.9620	106	1.0000
19	0.7509	41	0.8536	63	0.9169	85	0.9639		
20	0.7573	42	0.8570	64	0.9193	86	0.9658		
21	0.7635	43	0.8604	65	0.9217	87	0.9676		
22	0.7695	44	0.8637	66	0.9241	88	0.9695		

Table 3-4. A look-up table storing values of $\cos \phi_j$ for specular reflection with $I_p = 255$, $r + r_0 = 1.5$, $k_s = 0.7$ and $n = 8$.

j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$	j	$\cos \phi_j$
1	0.5502	25	0.8228	49	0.8950	73	0.9407	97	0.9748
2	0.6000	26	0.8269	50	0.8973	74	0.9423	98	0.9760
3	0.6312	27	0.8308	51	0.8995	75	0.9439	99	0.9773
4	0.6544	28	0.8345	52	0.9017	76	0.9455	100	0.9785
5	0.6729	29	0.8382	53	0.9038	77	0.9470	101	0.9797
6	0.6884	30	0.8418	54	0.9060	78	0.9486	102	0.9809
7	0.7018	31	0.8452	55	0.9080	79	0.9501	103	0.9821
8	0.7136	32	0.8486	56	0.9101	80	0.9516	104	0.9833
9	0.7242	33	0.8519	57	0.9121	81	0.9531	105	0.9845
10	0.7338	34	0.8550	58	0.9141	82	0.9545	106	0.9856
11	0.7426	35	0.8582	59	0.9160	83	0.9560	107	0.9868
12	0.7507	36	0.8612	60	0.9180	84	0.9574	108	0.9879
13	0.7582	37	0.8641	61	0.9199	85	0.9588	109	0.9891
14	0.7653	38	0.8670	62	0.9217	86	0.9602	110	0.9902
15	0.7719	39	0.8698	63	0.9236	87	0.9616	111	0.9913
16	0.7782	40	0.8726	64	0.9254	88	0.9630	112	0.9925
17	0.7841	41	0.8753	65	0.9272	89	0.9643	113	0.9936
18	0.7897	42	0.8779	66	0.9290	90	0.9657	114	0.9946
19	0.7951	43	0.8805	67	0.9307	91	0.9670	115	0.9957
20	0.8002	44	0.8831	68	0.9324	92	0.9683	116	0.9968
21	0.8051	45	0.8855	69	0.9341	93	0.9697	117	0.9979
22	0.8098	46	0.8880	70	0.9358	94	0.9710	118	0.9989
23	0.8143	47	0.8904	71	0.9375	95	0.9722	119	1.0000
24	0.8186	48	0.8927	72	0.9391	96	0.9735		



Figure 3-4. An algebraic surface of polynomial function shaded with the Phong specular reflection model and parameters $r + r_0 = 1.5$, $k_s = 0.7$ and $n = 8$.

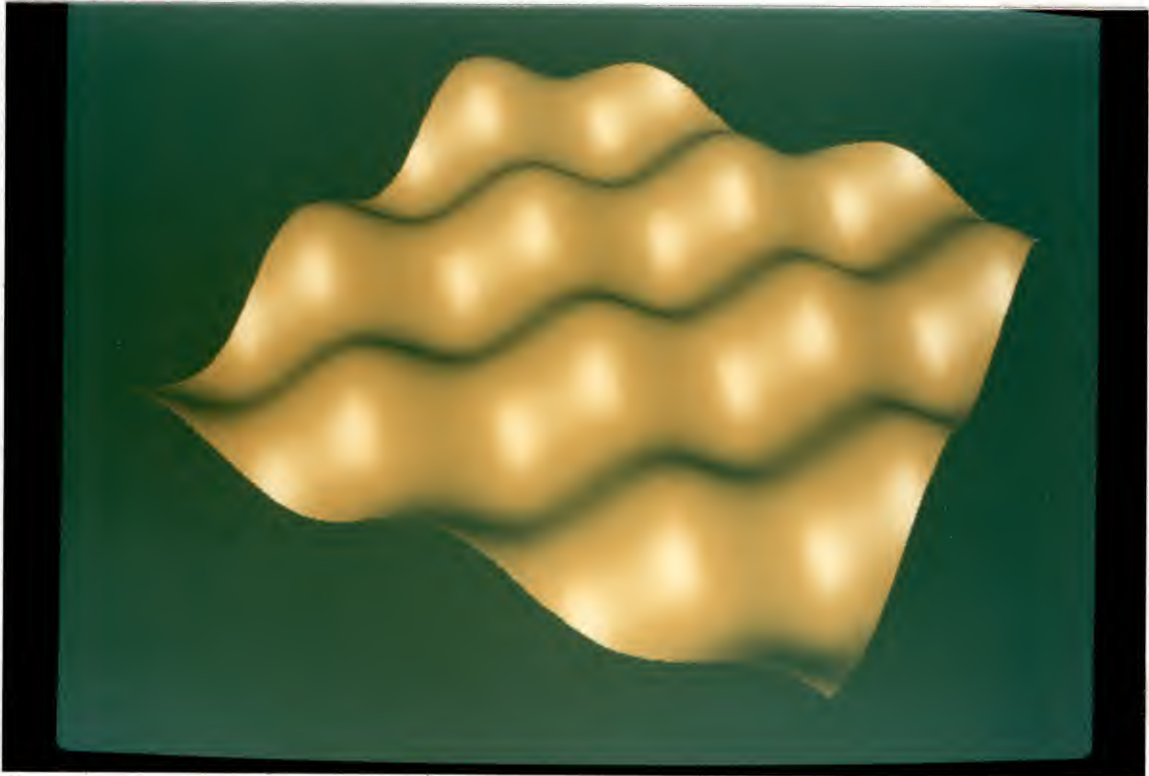


Figure 3-5 An algebraic surface of trigonometrical function shaded with the Phong model of specular reflection and parameters $r + r_0 = 1.2$, $k_s = 0.5$ and $n = 6$.

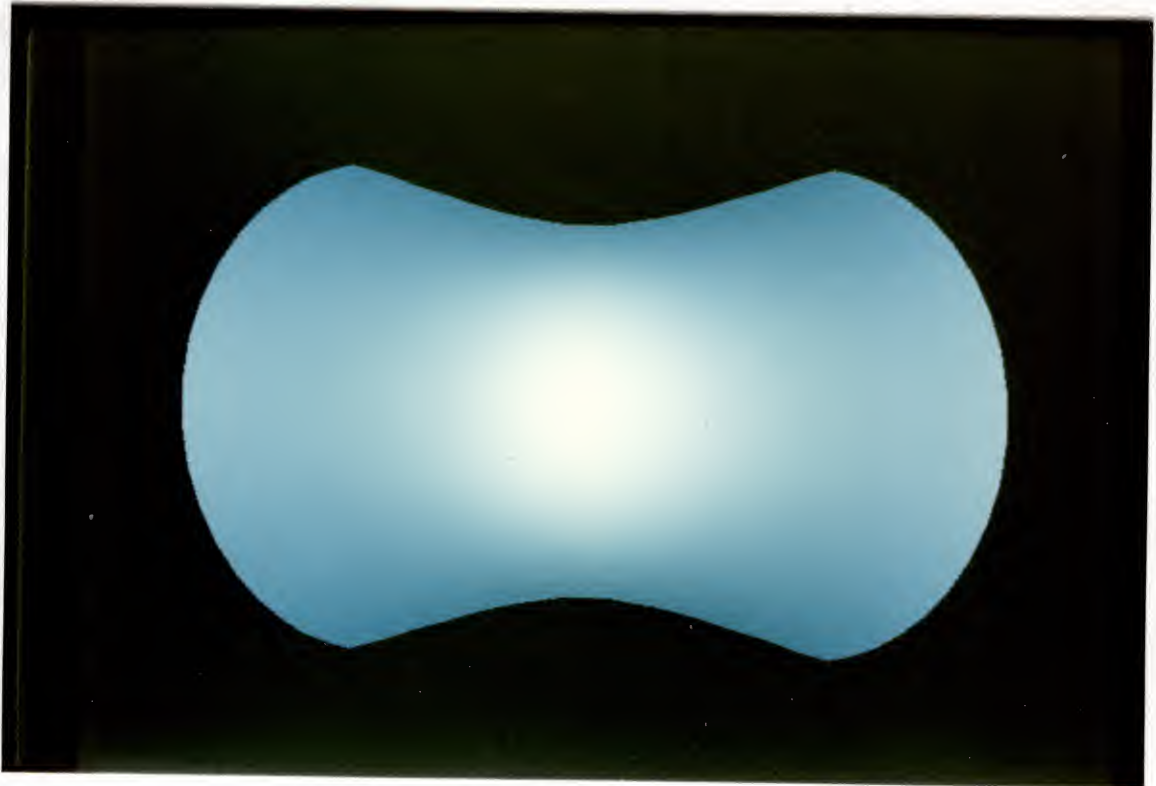


Figure 3-6. The algebraic surface of polynomial function in Figure 3-4 is shaded with the Torrance-Sparrow model of specular reflection here.

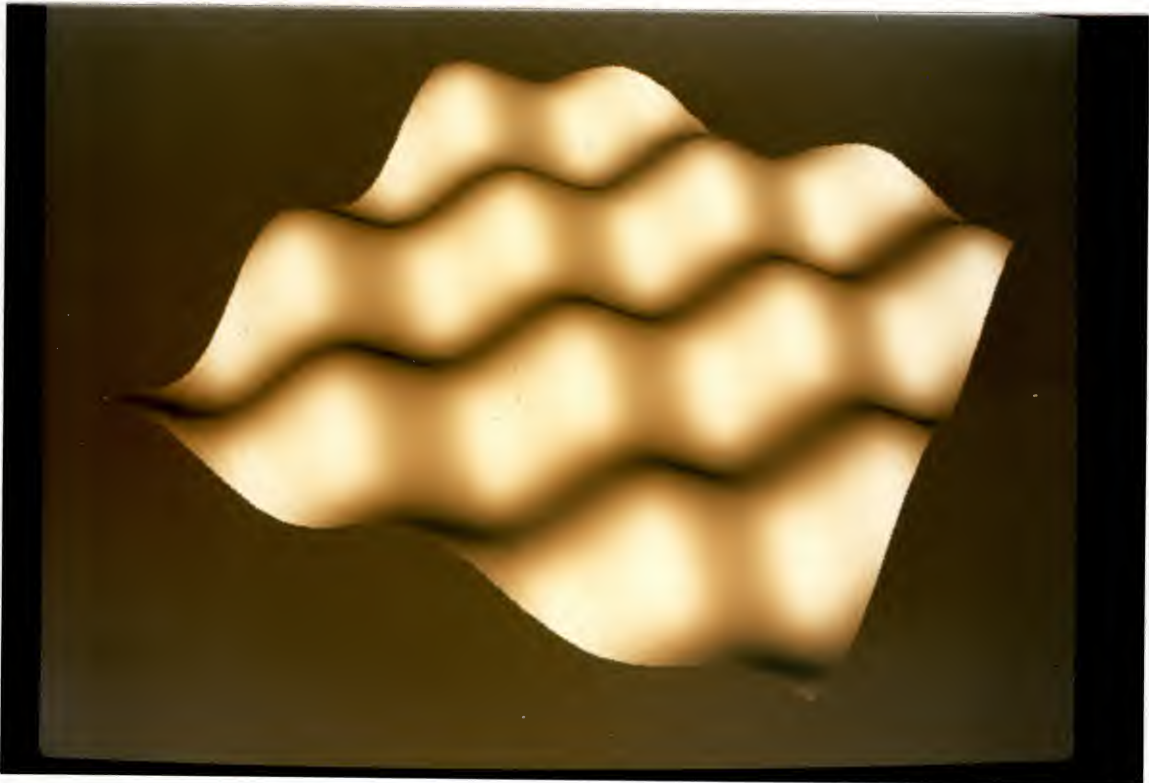


Figure 3-7. The algebraic surface of trigonometrical function in Figure 3-5 is shaded with the Torrance-Sparrow model of specular reflection here.

Techniques of Shading Continuity

In addition to the shading algorithm described above, some other properties can be applied for smoothly shaded images of objects defined by partially ordered surface data. An example is rendering a surface given by a complex variable function in accordance with its phase angle by specifying a particular color look-up table. This technique has been developed and improved.

A Color-Ring Look-up Table

Staudhammer [Stau75, Stau78] presented a color scale and used it in representation of a smooth shading surface image of multi-dimensional function. Such surfaces are generated from poles and zeros for the transfer function in control system analysis:

$$F(s) = \frac{k (s - Z_1) (s - Z_2) \dots (s - Z_{nz})}{(s - P_1) (s - P_2) \dots (s - P_{np})}$$

where $s = x + yi$, complex numbers; k is a constant; Z_i and P_i are the i^{th} zero and pole of a transfer function while np and nz are the number of zeros and poles. An algebraic surface of the transfer function is defined over the rectangle $(X_{\min}, X_{\max}, Y_{\min}, Y_{\max})$ with $N_x * N_y$ grid points. At each grid point the magnitude is measured along the Z -axis and a phase angle is calculated, which is then represented by a color scale. Representation of the phase angle by coloring should yield a way of uniquely tagging an angle (or a small range of angles) with a color, but must also have the closure property of a circle, in which angles can be measured smoothly: at the completion of a full circle one should be back to the same color. This can be accomplished by using three primary colors: linear combinations of 32 steps can go from green to red (for 0 to 120 degrees), red to blue (for 120 to 240 degrees) and blue to green (for 240 to 360 degrees). Hence binary colors can be used to describe any angle AND have a scale that has no discontinuities at the completion of a full

revolution. Therefore the magnitude of a complex variable function may be plotted as the vertical coordinate, and the phase angle as the color described above.

The Floating Perimeter Algorithm has been applied to render such surfaces as smoothly-shaded sheets including some improvements as well as a new rendering technique [Wang85b]. Both of these two approaches use the same design of "Color-Ring" look-up table but with two different number of intensity levels for each primary color, 32 and 64, and thus constitute two different total number of colors, 96 and 192, respectively, as shown in Figure 3-8.

The visibility determination on this complex variable function surface by the Floating Perimeter Algorithm is to apply the steps (I) to (III) of the algorithm to mapping the surface to the Viewing Coordinate System, classifying the viewing cases and enumerating the facets. The floating perimeter, i.e four one-dimensional arrays of boundaries, is now changed to a two-dimensional array with the height equal to the vertical resolution of the display, the width equal to the horizontal resolution of the display, and 1 bit deep. After that, the facets of the surface are displayed from the nearest one, i.e. with the smallest number, to the furthestmost one. All of the corresponding pixels of the first facet should be displayed and put flags in the boundary. Then, starting from the second processing facet, a check is made for a flag on each of the facet's corresponding pixels on the boundary. If the flag is present, the pixel is behind the other facet and needs not be shown, otherwise, the pixel of the facet is visible, display it and put a flag on it in the boundary. Therefore, in this front-to-back order with the checking-flag-display-or-not way, only the visible parts of the surface are displayed.

Painter's algorithm can also be applied to display the function surface but from the furthestmost facet to the nearest one, i.e. in the back-to-front order, so that all hidden parts on the back of the surface are displayed first and then covered by the visible portions in the front of the surface. This way of hidden-surface elimination does not require the boundary of the floating perimeter and flag-writing-and-checking. When portions of hidden surfaces

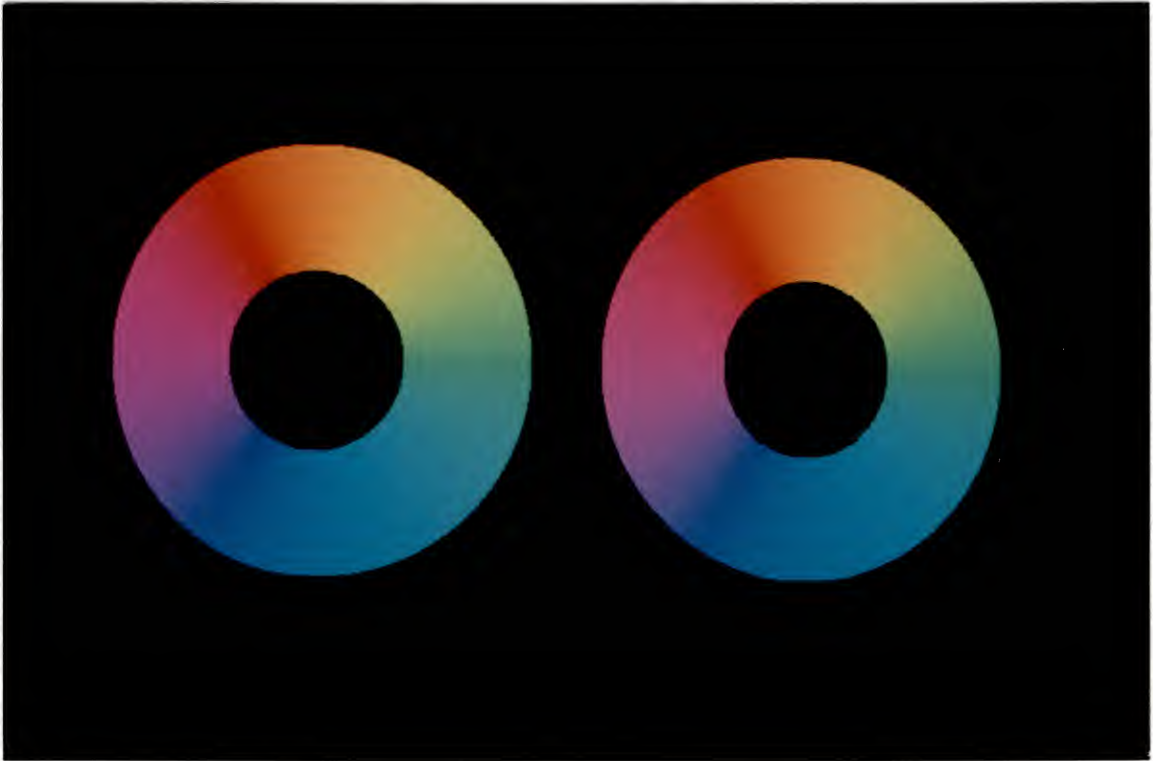


Figure 3-8. The color-ring look-up table with 96 colors (left) and 192 colors (right).

occupy only a small part of a scene, it is better to use the simple painter's algorithm. On the other hand, if the hidden parts of objects are substantial compared to the entire scene, shading only the visible parts of objects by using the boundary of floating perimeter and checking visibility flags should be more efficient. Especially in the computation-intensive event of shading each pixel individually by considering its surface normal, it is obvious that only the visible surfaces should be considered.

On displaying the visible parts of facets, each facet is split into two triangles and then each triangle is shaded with a fine phase angle color scale representation by the following procedures:

- (1) Let the phase angles at the vertices of a triangle be $pa[1]$, $pa[2]$, $pa[3]$.
- (2) The range of phase angles is $(-\pi, \pi)$. When three vertex phase angles are added and averaged, the surface tiling must be fine enough to produce a perceptively continuous color. The numerical discontinuity of $-\pi$ to π must be carefully handled.
- (3) Therefore, the method of making all average phase angles (colors) continuous is described as follows:

```

for (i = 1; i <= 3; i++)
{
    if (pa[i] <  $-\frac{\pi}{2}$ )
    {
        for (j = 1; j <= 3; j++)
        {
            if (i != j && (pa[j] - pa[i]) >  $\pi$ )
            {
                pa[i] +=  $2\pi$ ;
                goto next;
            }
        }
    }
    next: ;
}
avg_pa =  $\frac{pa[1] + pa[2] + pa[3]}{3}$ ;
if (avg_pa < 0) avg_pa +=  $2\pi$ ; .

```

(4) The color scale for the triangle is in the range (0,191) and is calculated as follows:

$$\text{avg_clr} = \frac{\text{avg} * 191}{2\pi}.$$

By using the Ring color look-up table and the Floating Perimeter Algorithm, a very smooth shading surface image of multi-dimensional function was displayed on a Silicon Graphics IRIS workstation (see Figure 3-9). An interactive technique for replacing any color in the look-up table to identify any given phase angle without additional computation had been developed and applied as shown in Figure 3-10. In addition, a sequence of functional surfaces was also generated on a Run-Length Decoder [Chen83]. The surfaces can be rotated in real time. Thus one can visualize an object from any direction and understand it more easily. Figure 3-11 and 3-12 show the surface observed from two different viewing angles.

Display Considerations

One of the concerns in shaded image display is the fidelity with which the shading information is reproduced on the screen. The shading intensity calculated precisely for each pixel from the above method indicates the light energy that should be generated at the spot and must be converted to one of the allowable intensity levels for the particular graphics system in use. Most of the systems are capable of displaying various intensity levels directly, while others can do only two levels for each pixel (on or off). The system used in this research supports 8 bits per pixel, that is, it has the capability to display 256 intensity levels for one primary color, red or green or blue, or 256 colors of their combinations. Therefore, a color look-up table with 256 entries has been created for the graphics system, in which each entry is composed of three elements--red, green and blue-- with one byte each. According to the color complexity of the objects, each of the colors in the entire scene can be specified as three color components with the desired intensity level respectively and

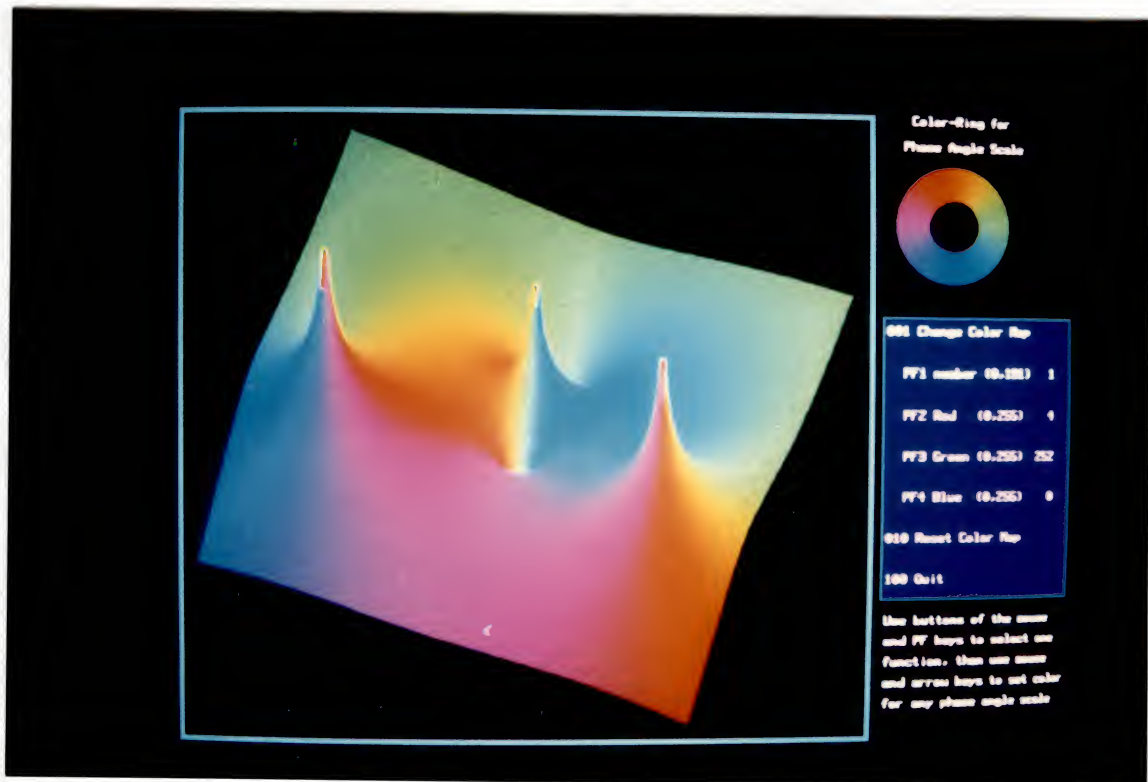


Figure 3-9. Multi-dimensional function display using a color scale.

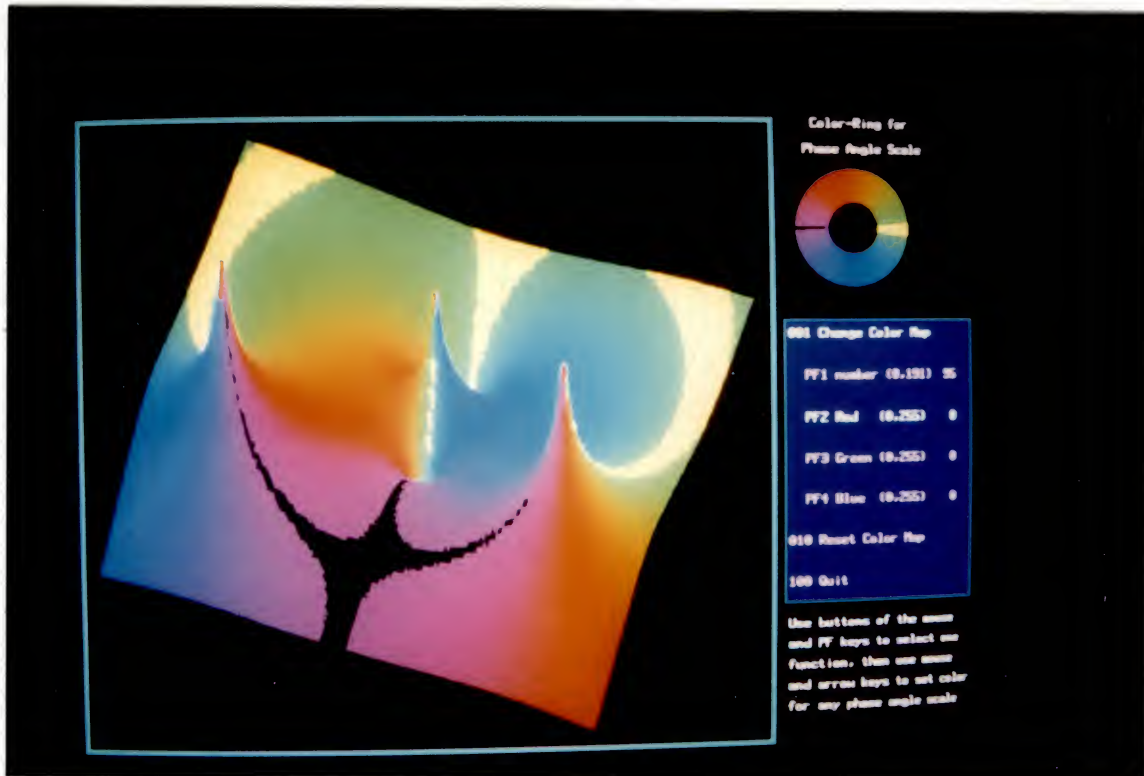


Figure 3-10. Multi-dimensional function display using a color scale with an interactive technique of color replacement for identifying any given phase angle.

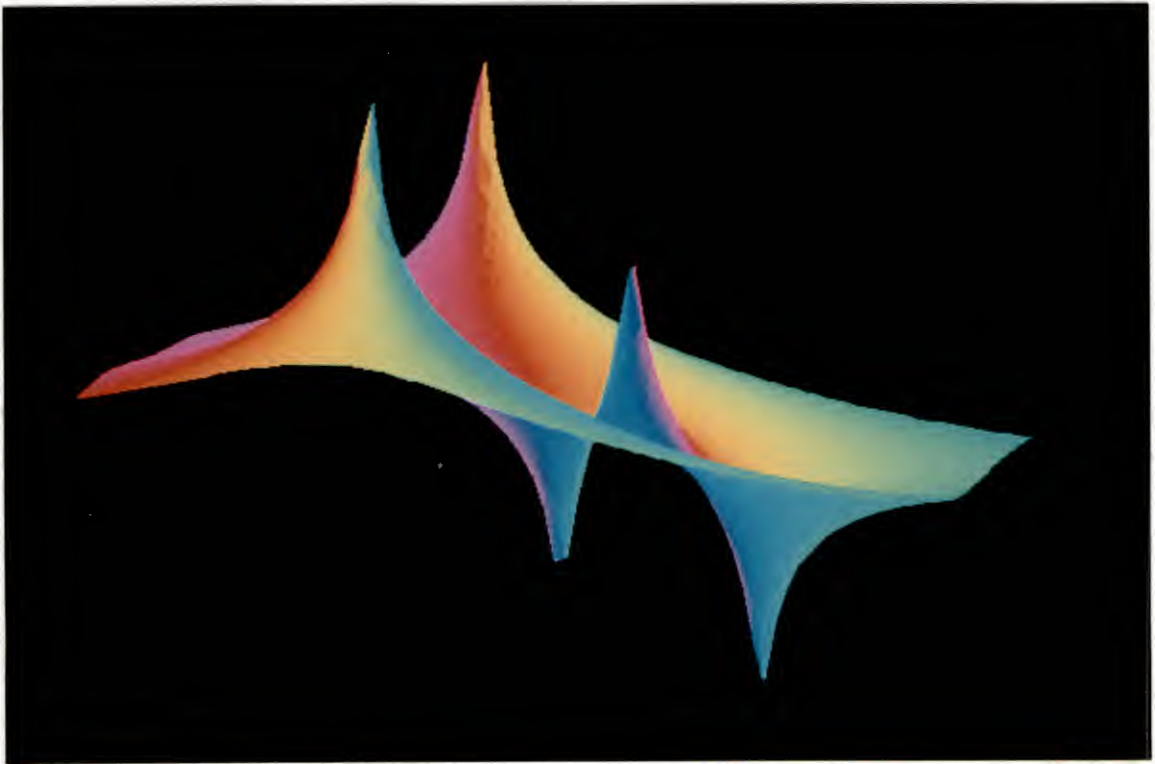


Figure 3-11. Multi-dimensional function surface with three poles and two zeros observed from a viewing angle.

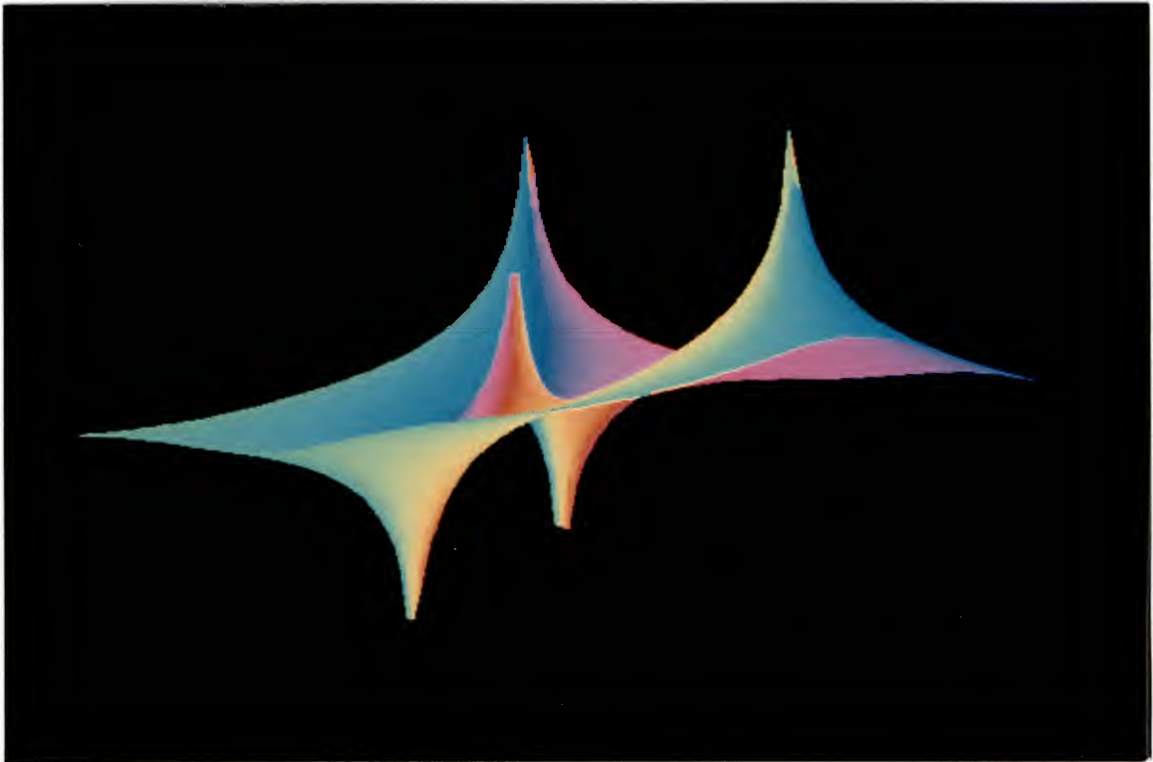


Figure 3-12. Multi-dimensional function surface with three poles and two zeros observed from a different viewing angle from that of Figure 3-11.

then be generated in the color look-up table. Here, the total number of intensity levels for each color component might be reduced from 256 to 128, 86, 64, ... or 8 because of the limitation of available number of colors on the system. In case of a graphics system supporting 24 bits per pixel, i.e. 8 bits or 256 intensity levels for each color component, more than 16 million colors can be displayed at a time and thus all natural colors seems to be able to be reproduced on the display device as long as the screen can accommodate them.

However, the shading intensity values calculated above do not have a linear relationship with the intensity values actually shown on the display for shading properly. That is because some distortions often occur due to non-linearities of the display introduced by the device hardware in the intensity-control circuit electronics and the CRT phosphors, which can distort the energy actually emitted at a pixel. To combat these effects, the shading calculation must compensate for the hardware effects. This task is easily accomplished with a compensation table [Catm79] to adjust the intensity value just before digital-to-analog conversion: given a calculated shading intensity I , we find in a table indexed by I a value as the new intensity for sending to the display. The table can be built by making photometric measurements of the light emitted by the display for each different intensity value passed to it.

Another non-linearity is that the human eye responds roughly to the logarithm of the incoming energy, hence, the intensity levels should be spaced logarithmically rather than linearly. The approach is to calculate the corresponding intensity values which in fact can be considered together in the above compensation table. Use of the look-up table in this general manner is called gamma correction. This has been discussed in detail as an important part of the calibration of a color monitor [Cowa83, Cowa85], and it is commonly used to correct for the non-linearities of the chosen display device [Ware88]. Some raster displays also include gamma correction as one of their features.

The non-linear intensity distortions also occur on film as well as on the display. The proper recording of colors on film is much too intricate to detail here but is discussed in a work by Hunt [Hunt75]. Therefore, it is worth noting that any picture recorded on a film, as any illustration used here, is only an approximation of an image on the display.

CHAPTER 4 ANTI-ALIASING

Introduction

The work of this dissertation is concerned with devising procedures to render images of three-dimensional objects on an interactive raster display. Raster displays are basically adaptations of the popular color television screen, driven from a computer interface. The television image is produced by a scanning electron beam as it pans across the display surface. This scanning pattern consists of a set of closely spaced horizontal lines, each the same length, and spaced evenly from the top to the bottom of the display screen. A light spot is produced as the electron beam hits the display surface which is coated with a light-emissive material, called a phosphor. The coloring of the light emitted (its temporal-spectral content) is a function of the phosphor material and is proportional (but not linearly proportional) to the electron beam energy. As the electron beam moves its intensity is modulated; in turn the light emitted from the spot where the electron beam hits the phosphor will make up parts of the display image. Thus display images will be a set of light spots which in a sense sample the space that is depicted by the image that is to be displayed.

As long as the phosphor surface is uniform, as it is with a black/white display tube, the monochrome image can be best described as a set of lines drawn on the display surface. Hence the display is a set of lines with potentially infinite resolution in the horizontal direction, but with a vertical resolution given by the number of scan lines employed. Thus horizontal resolution and vertical resolution are different and different algorithmic considerations apply for producing image smoothing in the two directions.

The display of colored image is accomplished with exciting different phosphor materials for the production of differently colored light spots. All visible colors can be

produced from a set of primary colors. The gamut of colors achievable using a given set of primary colors is those colors that can be made from that set [Ston88]. A gamut can be characterized by a linear combination of the primary colors. Clearly if only two primary colors are used (say Red and Green) only various shades of red, fading into yellows and light greens to dark greens, can be produced. Almost all of the visible colors can be produced from a set of Red, Green and Blue primaries. The particular colorings of these primaries are standardized for broadcast use: they are the NTSC color primaries RED, GREEN and BLUE. Thus broadcast colors are standardized for reproduction on receiving equipment. We note that since the standardization of these colors 'better' materials have been found which allows a wider color gamut to be reproduced: the Mitsubishi color primaries (sold in that company's TV tubes) allow some 15% more colors than the standard NTSC primaries. However, standard broadcasts do not contain those colors. Nevertheless, these wider gamuts may be useful in computer-controlled displays.

To produce the various colors on the display screen, three different phosphors are used, the three primary colors. Three electron beams are used, each of which is constrained so it can excite only one of the phosphors. This constraint is accomplished by placing the three electron beam source so each of them can 'see' only one set of the phosphor targets through a shield containing a set of holes. This shield is called the shadow mask. The display screen is thus effectively three images of the mask, spaced as a set of equilateral triangles which cover the entire display surface uniformly. Hence the display screen is a set of interdigitated points each with three color dots making up a primary-color triplet. It is this triplet that is referred to as a picture element, a pixel. The spacing of these points is approximately the same in each the vertical and the horizontal directions and the arrangement of dots is referred to as a raster; the image produced on these is a raster image. For interactive displays the resolution of the display screen is typically about 1000 lines by about 1000 pixels. The algorithms developed in this chapter are aimed at such display resolutions.

This dissertation is concerned with methods for producing good-looking images on such displays. The origin of data describing these images is the location and surface properties of items in three space. Essentially a scene described in three-space must be sampled and each of the pixels that make up the picture must be calculated as to the amount of red/green/blue light that is to be generated at the respective pixel locations. Hence the images which are displayed are discretely sampled values of a potentially infinitely, finely, detailed object space. The process of sampling can introduce errors in the image; these errors are collectively known as aliasing effects. We shall look at some procedures to minimize or avoid these effects for specific types of images. The processes for combatting the aliasing effects are known as "anti-aliasing."

As mentioned above, the images on the raster display are discrete representations of continuous objects in geometric space. The continuous images must be sampled at regularly spaced locations in order to be displayed on a raster device. This process of converting a continuous signal into a discrete signal is called sampling. The sampling theorem [Oppe75] states that the input signal can be reconstructed exactly from its samples only if it is bandlimited to some maximum frequency which is less than one-half the sampling rate. This frequency limit is called the Nyquist limit. Any frequency component of the signal that is higher than this limit will be reconstructed as a lower frequency coarse signal.

The spatial-sampling rate of a synthetic image is fixed by the resolution of the raster device. Since a continuous image of an object must be discretely and regularly space-sampled, the spacing of the samples limits the amount of detail that can be represented. Any attempt to display finer detail will result in the introduction of an unwanted coarse signal. This unwanted signal is called an alias and this process is known as aliasing.

The effects of aliasing in two dimensions are common in computer graphics, including "jaggy," "staircased" line edges, Moire patterns in textures, jagged highlights and, in dynamic displays, strobing of a fast moving object, uneven crawling of a slowly moving object and scintillation on very small moving objects appearing and disappearing between

pixels. Therefore, the common appearance of "jaggies" on a raster image is one kind of alias. It is due to the fact that lines, polygon edges, color boundaries, etc. of an object's image are continuous, whereas a raster display shows only regularly spaced samples.

Anti-aliasing is the process of preventing aliasing. To perform anti-aliasing, the continuous signal must be filtered before sampling to eliminate all detail which is too small. That is, to reduce the effects of aliasing in the final image, the obvious method is to filter the image to remove or reduce the high-frequency components before sampling. An ideal filter for anti-aliasing should produce the best looking pictures which, by definition, involves how the human eye/brain system extracts information from an image. An early vision system has been modeled [Stoc72] and guidelines for synthesizing images, including accommodating color-defective observers, is also presented [Meye88]. Usually an image is linearly filtered by convolving it with a kernel or point-spread function. A filter with a smooth, all positive, finite extent point-spread function and smooth, finite frequency response would seem to be superior for processing images for human viewing.

The filtering and sampling process of an image yields an anti-aliasing integral for the samples representing the image. Therefore, all anti-aliasing algorithms attempt to solve or approximate the anti-aliasing integral at each pixel. Algorithms vary in how this calculation is done and what point-spread function is used. Basically, two types of anti-aliasing algorithms are used in computer graphics. They are directly related to the two types of visible-line/surface algorithms in use, object-space and image-space, or continuous and point-sampling. Thus anti-aliasing algorithms can be classified as analytic or discrete. Analytic algorithms are continuous which solve the anti-aliasing integral directly. Typically, these are limited to simple primitives and point-spread functions so this integral can be easily solved to yield the filtered sample values. On the other hand, discrete algorithms only consider the image at regularly spaced sample points. For instance, super-sampling, adaptive super-sampling and stochastic, or jittered, sampling are discrete approaches to

alleviating the aliasing problems. These techniques generally rely on taking several point samples and using them to approximate the continuous image in the area of interest.

Many analytic and discrete anti-aliasing algorithms have been developed in over a decade. Some approaches are useful only for certain kinds of images, whereas most methods can be used in general. This chapter first reviews existing anti-aliasing algorithms and then presents some area-sampling methods for rendering primitive objects, such as lines, edges of polygons, circles and silhouettes of spheres, and also for generating smoothed images of solid objects with partially ordered surface data. For these anti-aliasing techniques, intensity sampling look-up tables can be produced and then used to superimpose on visible-line or visible-surface images as a rapid post process of anti-aliasing. "Post process" refers to the fact that these algorithms are applied after visibility and coloring have been calculated (usually with a spatial sampling procedure.)

Literature Review

The earliest analytic algorithm with continuous anti-aliasing was that of Catmull [Catm78], which used polygons as primitives for 3-D scenes and adopted an area sampling filter point-spread function to render the image. By considering all visible polygons at each pixel, the algorithm uses an area sampling filter over a unit square centered on the current pixel and then approximates the anti-aliasing integral. This filter function and choice of image primitives makes it particularly easy to calculate the anti-aliasing integral, since it reduces to the product of the areas of all visible primitives and their corresponding intensities at each single pixel. Although this integral is easy to calculate and the final image is much better than no filtering at all, the high-frequency cutoff of this filter is fairly poor and some aliasing artifacts still remain.

Feibush et al. [Feib80] presented filters with radially symmetric, small extent point-spread functions and generated a two-dimensional look-up table to hold precomputed integrals of the filter function over various triangular domains. The filter is also weighted

by the Gaussian function for producing more smoothed images. Since all the integration was precomputed, there is no run-time performance penalty to be considered regardless of the complexity of the filter. The filter can be changed just by using a different look-up table. Therefore, this table driven algorithm made a significant contribution for calculating the anti-aliasing integral. The algorithm can eliminate almost all aliasing artifacts.

A more simplified anti-aliasing algorithm, proposed by Turkowski [Turk82], uses a much smaller one-dimensional look-up table. The algorithm calculates the perpendicular point-line distance, i.e. distance from the inferred edge to the pixel center, to evaluate the two-dimensional anti-aliasing convolution, and then uses this distance as the index of a function to the final intensity value of the destination pixel in the look-up table. Many aspects of the anti-aliasing calculations were performed by using cordic-arithmetic methods. Edges of polygons can be anti-aliased by this algorithm, but vertices can only be approximated.

Norton et al. [Nort82] use an object-space anti-aliasing method to filter the input signal according to the rate at which it is to be reproduced. The higher frequency terms, which are usually truncated, are selectively dampened by the algorithm according to a power series approximation of a box filter. The result is a continuous spatial and temporal anti-aliasing technique which can eliminate certain types of aliasing artifacts very effectively.

Character fonts displayed on raster devices need not only be readable but pleasant to view as well. Warnock [Warn80], Kajiya and Ullner [Kaji81] presented filtering and sampling methods for effectively generating a wide variety of high quality fonts of different sizes and different orientations automatically.

Super-sampling, one of the first discrete techniques developed for approximating the anti-aliasing integral, involves using more than one regularly spaced sample per pixel. It calculates the aliased image at a higher resolution and then generates the lower resolution final image by using uniform or weighted averaging to obtain the pixel attributes at the lower resolution. The algorithm can be very expensive if a large number of samples are

used to form each output pixel. It can reduce aliasing, but it does not eliminate it. No matter how many samples are used, there are still frequencies that will alias. These undesirable visual effects in terms of aliasing phenomena inherent in sampled signals was explained and described earlier by Crow [Crow77]. Later he presented a comparison of prefiltering (continuous) and super-sampling anti-aliasing methods with a set of computer-generated images [Crow81a].

Adaptive super-sampling, first described for ray-tracing by Whitted [Whit80], is an attempt to reduce the number of samples required in sections of the image which are relatively uniform. For each pixel on the screen, a ray of infinitesimal width is traced from the specified eyepoint through the pixel, into object space. Samples are taken at the corners of the square region defining a pixel and intensities are also calculated. When the intensity variance for rays traced at the corners of a pixel is too large, additional rays are traced for minimizing the aliasing effect of objects. Therefore, this algorithm saves a great deal of calculation in uniform parts of the image and concentrates the sampling in the areas where it is needed most. Aliasing in highlighting was also addressed in the paper.

A method of displaying anti-aliased lines on a raster display by painting with an anti-aliased brush was also presented by Whitted [Whit83]. The anti-aliasing calculation is performed once for the brush itself and thereafter only a trivial additional operation is needed for each pixel through which the brush is dragged to yield an anti-aliased line. Lines can be curved as well as straight, but there are a few constraints on the size, shape, and attributes of the brush. Although the method is simple to implement, it requires a large amount of frame memory.

Algorithms for detection and smoothing of edges were presented [Bloo83]. A staircase edge, one common aliasing artifact on the raster display, may be smoothed by inferring, from the set of vertical and horizontal segments which form the staircase, an approximation to the original edge with a precision beyond that of the raster. The filtering of an image is

also done in accordance with the inferred edge. The algorithm operates in linear time with respect to the length of an edge and can produce convincing results.

For all anti-aliasing techniques, the solution proposed by Fujimoto and Iwata [Fuji83] is one of the fastest and simplest methods. Their algorithm involves the adoption of a filter function that establishes linear relations between the intensity of filtered pixels and some control variables defining the edge geometry. The algorithm flow chart shows that each pixel intensity is inversely proportional to the distance from the vector's center line and uses an incremental calculation to generate both the pixel's intensity and its position. However, it also shows that the calculation of the intensity for a unit length of the vector does not account for any given vector slope and thus the vector produced can not have the property of constant-intensity: the intensity per unit length is not constant.

Some accurate and efficient area sampling techniques for rendering some primitive objects, such as lines, edges of polygons, circles, and silhouettes of spheres have been developed [Wang84] recently. Techniques for smoothing silhouettes of circles and spheres are based on those for anti-aliasing lines and edges of polygons, respectively. They all utilize the symmetry properties of objects and floating point operations to precisely calculate the covered pixels areas for modulating the intensity in order to anti-alias the objects exactly. An anti-aliased line is produced according to its slope and end points of real coordinates and thus is a constant-intensity line. In the next section of this chapter, these area sampling algorithms are described in detail.

The A-buffer (anti-aliased, area-averaged, accumulation buffer) algorithm of Carpenter [Carp84] combines different anti-aliasing methods in actual implementations. It performs continuous anti-aliasing at each pixel for the simple case, but uses super-sampling when much aliasing is detected.

The stratified sampling algorithm, introduced by Lee et al. [Lee85], is an enhanced adaptive super-sampling technique where samples are placed and weighted based on the estimated local variance of the signal near the sample. Thus, the need to estimate the local

variance of the image limits the usefulness of this algorithm, although the method has been optimized for various types of images through the use of statistical testing.

Since regularly spaced sampling produces regularly spaced artifacts, perturbing the sampling locations slightly breaks up the regularity of the aliasing artifacts. Stochastic sampling [Cook86] applies a small random phase shift to each sample using a 2-D Poisson disk distribution and shows that most aliasing artifacts are not an inherent part of point sampling, but a consequence of using regularly spaced samples. It jitters alias using random noise. Thus perceptibly better images are produced because it is much easier for the human eye to ignore random noise than a regular pattern.

Cook et al. [Cook87] presented the REYES rendering system which utilizes stochastic point sampling with a z buffer. All objects are reduced to common world-space geometric entities called micropolygons which are then transformed to screen space. Stochastically sampling these micropolygons within each pixel, a weighted average of intensities at the subpixels is produced and a high-quality anti-aliased image can thus be produced.

The problems of temporal anti-aliasing have been solved in several ways. Correct temporal anti-aliasing requires solving all visibility problems as a function of time, then filtering the results. One of the first techniques was presented by Korein and Badler [Kore83], which uses a simple continuous one-dimensional anti-aliasing algorithm in the temporal dimensions while super-sampling in the spatial dimensions. Grant [Gran85] extends the results of Feibush et al. [Feib80] from a two-dimensional look-up table algorithm to a three-dimensional look-up table algorithm for combined spatial and temporal anti-aliasing by using radially symmetric filters.

Some approaches blur images after calculating visibility at one sample point in time. This technique is commonly called motion blur. Catmull [Catm84] implements a motion blur approximation by shrinking the polygons in the direction of motion before visible surfaces are calculated. Max and Lerner [Max85] present a temporal anti-aliasing algorithm for raster and vector motion blur that produces masks suitable for $2\frac{1}{2}$ -dimensional

composition and utilize efficient one-dimensional blurring schemes to make the algorithm fast.

It is often assumed that the anti-aliasing process destroys useful visual information about object features, because methods for eliminating aliasing artifacts from synthetic images usually involve a filtering operation and thus reduce the subjective clarity of the images. However, Ferwerta and Greenberg [Ferw88] presented three experiments to examine the effects of anti-aliasing on visual information for object location and motion. The results of these experiments show that proper anti-aliasing can eliminate the spurious visual information produced by image filtering and allow the viewer's visual system to work effectively to the limits of their accuracy, i.e. to produce a precise representation of object location and a continuous representation of object motion. These results suggest that in designing imaging display systems, simply increasing the spatial and temporal addressability and resolution beyond limits set by the human's visual system will have negligible impact on image quality, but that effective use of anti-aliasing techniques, which take the perceptual processes into account, can allow all visual information about objects features to be presented with great fidelity.

Area Sampling Algorithms

On a raster display, rendering of anti-aliased images for smooth objects is usually accomplished by blurring their silhouettes. A properly adjusted digital raster display consists of regularly spaced dots which overlap by about one-half [Conr85]. The calculation of a proper anti-aliased edge or line would have to take into account the exact nature of the writing beam and the display pixel structure. The beam typically is not a round spot, but rather has an elliptical shape. However, quite acceptable results are obtained from a simple model of the display raster structure that considers the pixels as abutting squares rather than overlapping dots. Here we use that raster structure for developing an analytic anti-aliased approach for removing the aliasing effects from raster images of lines, polygon

edges, circles and sphere silhouettes. The algorithms use an area sampling process for each pixel that needs to be anti-aliased. The calculations are based on a distance between the pixel and the center of the line, edge, circle or sphere; in the case of a line or an edge, the slope is also used in the calculations. The partial pixel areas covered are calculated and the light intensities in each of the pixel parts are computed. The area sampling procedures are thus termed DISLOP_L, DISLOP_E, DISLOP_C and DISLOP_S for anti-aliasing line, polygon edge, circle and sphere silhouettes, respectively

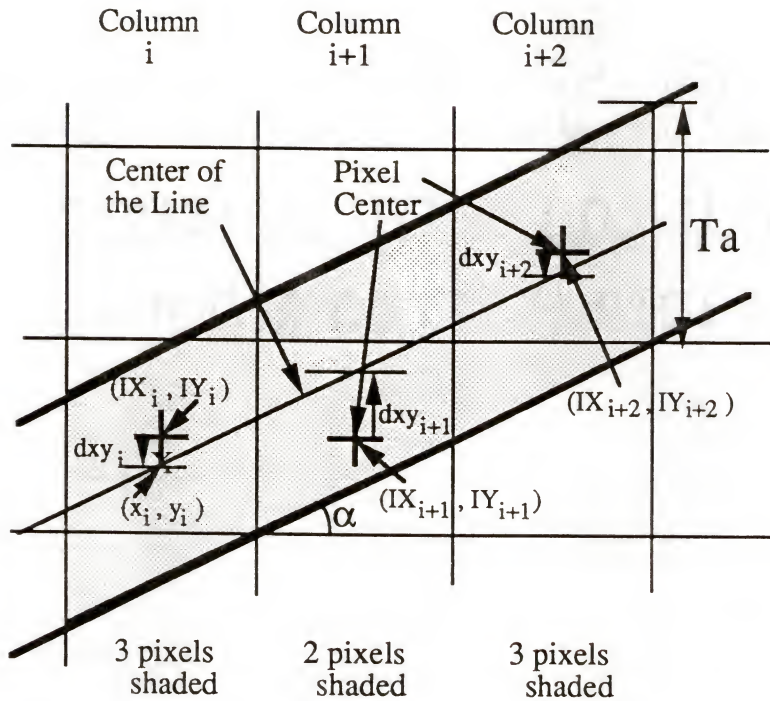
These algorithms utilize floating point operations for the precise calculation of areas in order to anti-alias the object's image accurately. Lines and circles are defined as a pixel thick instead of being infinitely thin. The anti-aliasing circle algorithm DISLOP_C actually takes advantage of the algorithm DISLOP_L for rendering line segments which constitute a circle, whereas the anti-aliasing algorithm DISLOP_S utilizes DISLOP_E to render the silhouette of a sphere as a set of short edges of a polygon which fits the sphere. Techniques and accuracies of these algorithms described as follows show that they indeed remove aliasing effects on the objects. In comparison with other anti-aliasing approaches, the results are quite amazing.

For generating smoothed images of solid objects with partially ordered surface data, these area sampling algorithms certainly can be applied. Furthermore, these anti-aliasing techniques can use intensity sampling look-up tables for a rapid post processing to remove aliasing effects from gridded or shaded surface images.

Algorithm DISLOP_L for Line Rendering

An anti-aliased line must be a constant-intensity line which means that the intensity per unit length is constant. For example, all lines 1-pixel wide and 10-pixels long with the same intensity but with different slopes should be displayed as lines in which the total light produced by the line is a constant and uniform per unit length along the line.

The algorithm DISLOP_L can generate constant-intensity anti-aliased lines with end points in real coordinates and with slopes in any octant of the image space. The slope of the line, m , and the vertical or horizontal distance, d_{xy} , between the pixel center and the center of a line in the same pixel square are two factors of this procedure which determine the number of pixels (1, 2 or 3) that the line covers in each column or each row and the percentage of area coverage of these pixels. Although a 2-D image space can be defined as eight octants, lines displayed in these octants can be classified as four types only, i.e. each line has a slope in one of four ranges- $[0,1]$, $(1,\infty)$, $(-\infty, -1)$, $[-1,0)$. Since all lines are defined as one-pixel thick and have the property of straightness, techniques for anti-aliasing these four types of lines are very similar. Therefore, it is necessary to show the detailed method for only one type of lines. Figure 4-1 shows the parameters for calculating the pixel intensities for an anti-aliased line in the slope range $[0,1]$. Details of the algorithm, DISLOP_L, are given in Figure 4-2 for smoothing the middle section of the line. It is obvious that the anti-aliased lines have to be smoothed at both ends as well as in the middle. However, depending on the location of the end point of a line in a pixel, nine cases of different pixels covered by the ends of a line can be classified. Figure 4-3 shows the nine possible cases and the start point of a line for case 1. The area sampling technique for smoothing this end of the line is to calculate all pixel area coverages in the first three columns, and then to multiply each pixel's coverage by the line's intensity in order to get the intensity for the pixel. Although the procedures for anti-aliasing ends of lines in each different case are a little different, the concept is always the same, i.e. merely 2-D geometric calculation for their corresponding pixels' area coverages. Smoothed lines produced by the anti-aliasing algorithm DISLOP_L are presented on the right of Figure 4-4, whereas the ordinary non-anti-aliased lines are on the left.



```

float  m, dxy, abs_dxy, Ta, dw, fcv, scv, sh,
mid_area[500],      /* the area of the pixel always covered by the line. */
up_area[500],       /* the area of the upper pixel might be covered by the line. */
low_area[500];      /* the area of the lower pixel might be covered by the line. */

m = tan (α);        /* the slope of the line. */
dxy = yi - IYi;     /* the vertical distance between the pixel center and
                    the center of the line in the same pixel square.
                    The range of dxy is (-0.5, 0.5). */

abs_dxy = fabs (dxy); /* the absolute value of dxy. */
Ta = 1.0 / cos (α);  /* the total area of the pixel covered by the line per column. */
dw = Ta - 1.0;       /* the difference between the width of a pixel, 1, and of the
                    line in the vertical direction. */

fcv = 0.5 * (m + dw); /* the first critical value used to judge the number of pixels
                    covered by the line in the current processing column. */

scv = abs_dxy + 0.5 * dw; /* the second critical value used to store the covered areas
                    and to judge the way for smoothing the line. */

sh = 1.0 / (2.0 * m); /* a variable used for calculating the area. */

```

Figure 4-1. A line with one-pixel wide and a slope in the first range, [0,1].

Assume the start point of the line is in the pixel (ixs, iys) and
the end point of the line is in the pixel (ixe, iye).

```

int    i, nm2, ix, iy;

    nm2 = ix - ixs - 1; /* number of columns passed by the middle section of the line. */
    i = 1; /* start from the column 1 instead of 0. */
    ix = ixs; iy = iys; /* initialize the x- & y-coordinates. */
loop:
    dxy = dxy + m;
    if (dxy > 0.5)
    {
        iy = iy + 1; /* the y-coordinate of the middle pixel. */
        dxy = dxy - 1.0;
    }
    ix = ix + 1; /* the x-coordinate of the ith column. */
    if (m == 0.0) /* the slope is 0. 1 or 2 pixels are covered by the line. */
    {
        up_area[i] = abs_dxy;
        low_area[i] = 0.0;
    }
    else
    {
        if (abs_dxy > fcx) /* 2 pixels are covered by the line in the ith column. */
        {
            up_area[i] = scx;
            low_area[i] = 0.0;
        }
        else /* 3 pixels are covered by the line in the ith column. */
        {
            if (scx > 0.5 * m)
            {
                up_area[i] = scx;
                low_area[i] = sh * ((fcx - abs_dxy) ** 2);
            }
            else
            {
                up_area[i] = sh * ((abs_dxy + fcx) ** 2);
                low_area[i] = sh * ((fcx - abs_dxy) ** 2);
            }
        }
    }
    if (dxy < 0.0)
    {
        temp = up_area[i];
        up_area[i] = low_area[i];
        low_area[i] = temp;
    }
    mid_area[i] = Ta - up_area[i] - low_area[i];
    if (++i <= nm2) goto loop;

```

Figure 4-2. The algorithm DISLOP_L calculates the pixel areas covered by the middle section of a line with a slope in the first range, [0,1]. See Figure 4-1.

(a)

Valid Invalid	*	*		*	*	*	*	*	*	*	*	*	*	*	*	*
			*	*	*		*	*				*		*	*	*
ixsu	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0
iysu	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
ixsd	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
iysd	-1	0	-1	0	-1	0	-1	0	-1	0	-1	0	-1	0	-1	0
Case No.	1	2					3			4	5	7	8		6	9

(b)

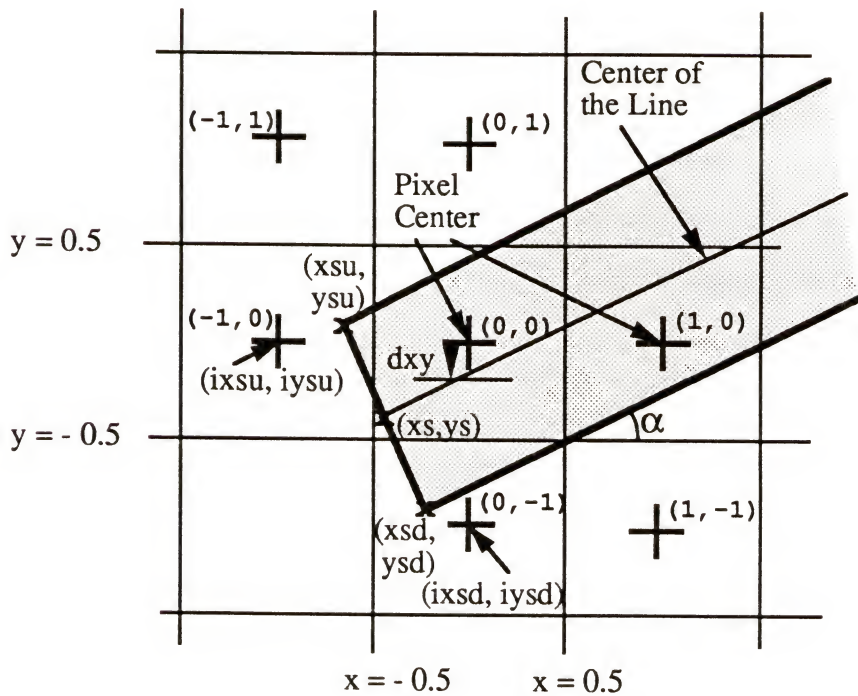


Figure 4-3. (a) Nine cases depending on the location of the start point. (b) The start point of a line for case 1.

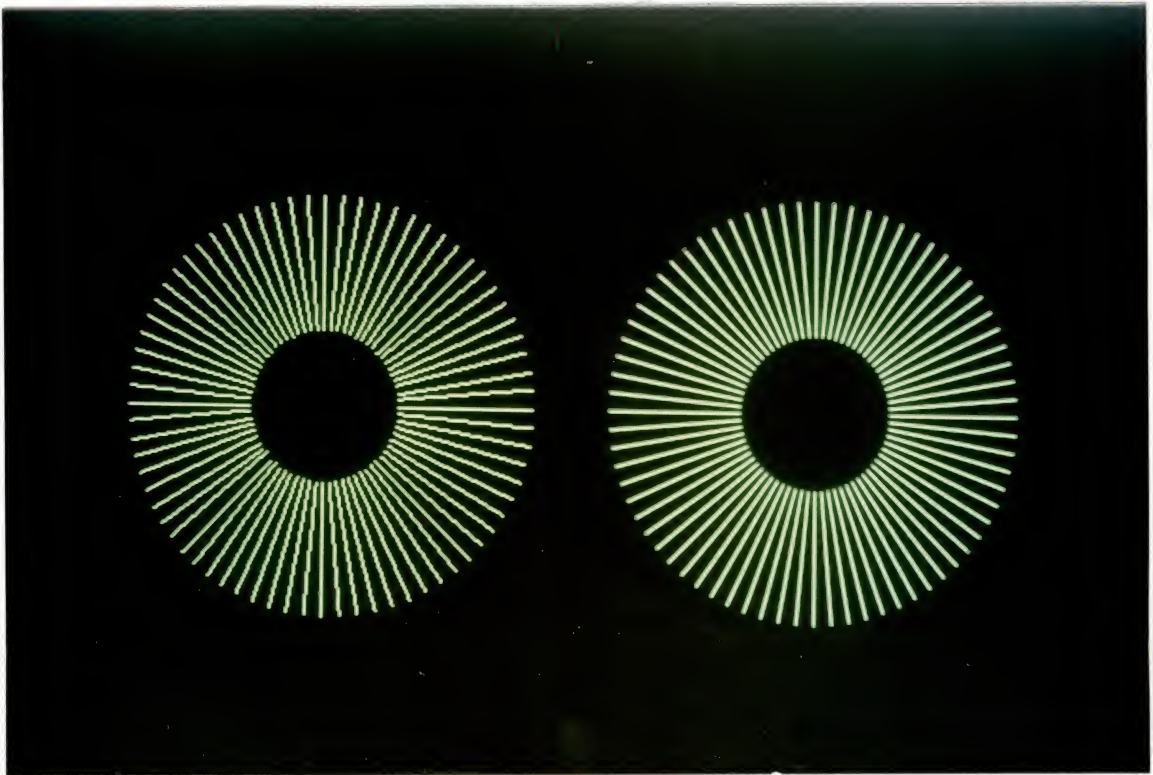


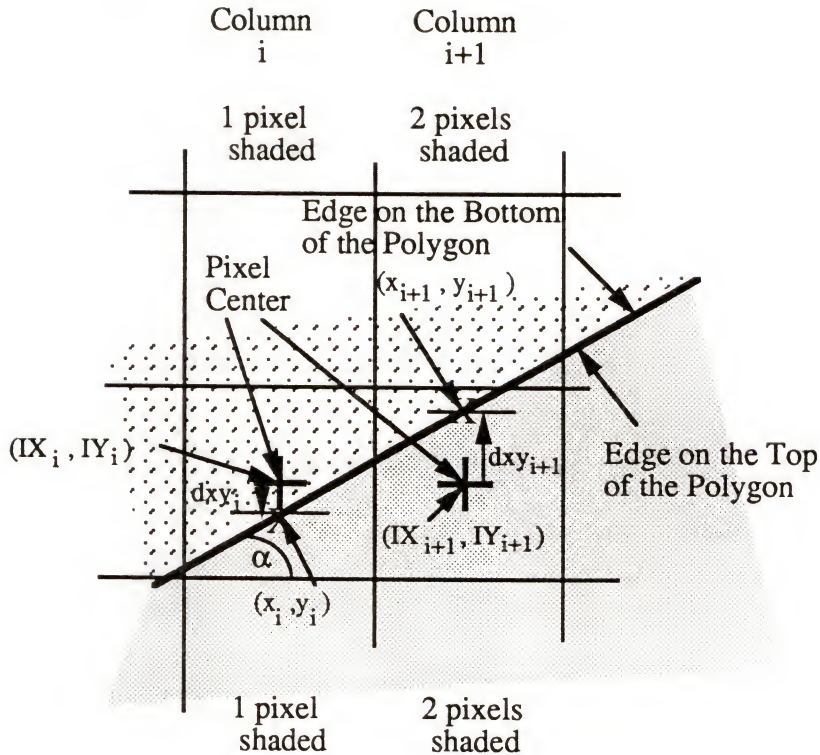
Figure 4-4. Anti-aliased lines (right) using the algorithm DISLOP_L and non-anti-aliased lines (left).

Algorithm DISLOP_E for Edge Rendering

Even though an anti-aliased line presented above can be treated as a "skinny polygon," the technique for smoothing most of polygon edges is quite different from that for a line. Therefore, the area sampling algorithm described above for rendering lines, DISLOP_L, needs to be modified for rendering edges of polygons. The significant differences between these two rendering methods can be stated as the following three points:

- (1) The total intensity of pixel areas covered by a line in one column, or in one row, which depends on the slope of the line, is a constant, but it is unpredictable for any polygon with a shape different from that of a "skinny polygon" as a line.
- (2) An edge divides each of the pixels through which it passes into two parts. We will use a flag to indicate which part is inside the polygon and thus needs to be considered in calculating the smoothed edge using the polygon's shading.
- (3) The ends of an edge of a polygon are relevant to those of the polygon's other edges which share common vertices, whereas the ends of a line are independent. Therefore, calculation of the pixel coverage around the ends of an edge should be treated separately.

In Figure 4-5, a polygon edge with a slope in the first range, $[0,1]$, divides most of passing pixels into two halves, and then the anti-aliasing edge algorithm DISLOP_E shows how to calculate pixel areas on both sides divided by the edge in Figure 4-6. For determining which side of passed pixels covered by a polygon, it is essential to set two flags, such as 0 and 1, and then assign a proper flag to each edge of the polygon. For instance, an edge with the slope in the first range, $[0,1]$, and on the top of polygon can be assigned flag 1, whereas flag 0 is for other edges with slopes in the same range but on the bottom of polygons. The flag assignments for edges with slope in other three ranges as well as in the first range are better described as shown in Figure 4-7, in which most of edges have a jagged side in addition to the other smoothed side. The jagged sides are actually inside polygons and will disappear after the edges being used to superimpose on



```

float    m, dxy, abs_dxy, cv, AA, T1, T2,
mid_area[500],    /* the area of the pixel, which is always passed by the edge,
                    covered by the polygon. */
up_area[500],    /* the area of the upper pixel, which might be passed by the edge,
                    covered by the polygon. */
low_area[500],    /* the area of the lower pixel, which might be passed by the edge,
                    covered by the polygon. */
flag;            /* indicates whether the edge with a slope in the first range, [0,1],
                    is on the top or the bottom of the polygon. if the value is 1,
                    then the edge is on the top; otherwise on the bottom. */

m = tan (alpha);    /* the slope of the edge of the polygon. */
dxy = y_i - IY_i;    /* the vertical distance between the pixel center and
                    the edge in the same pixel square.
                    The range of dxy is (-0.5,0.5). */
abs_dxy = fabs (dxy);    /* the absolute value of dxy. */
cv = abs_dxy + 0.5 * (m - 1);    /* the critical value used to judge the number of pixels
                    passed by the edge in the current processing column. */
AA = (cv ** 2) / (2.0 * m);    /* AA, T1 and T2 are variables used for calculation of
T1 = 1.5 - abs_dxy;    /* two edge-passed pixel areas covered by the polygon
T2 = 0.5 + abs_dxy;    /* with the edge on the top or on the bottom.

```

Figure 4-5. A polygon edge divides most of passing pixels into two halves. The edge used here has a slope in the first range, $[0,1]$.

Assume the start point of the edge is in the pixel (ixs, iys) and the end point in (ixe, iye).

```

int    i, nm2, ix, iy;

    nm2 = ix - ixs - 1; /* number of columns passed by the middle section of the edge. */
    i = 1;              /* start from the column 1 instead of 0. */
    ix = ixs; iy = iys; /* initialize the x & y-coordinates. */
loop:
    dxy = dxy + m;
    if (dy > 0.5)
    {
        iy = iy + 1; /* the y-coordinate of the middle pixel. */
        dxy = dxy - 1.0;
    }
    ix = ix + 1; /* the x-coordinate of the ith column. */
    if (m == 0.0 || cv < 0.0) /* only one pixel is covered by the edge . */
    {
        if (top_bottom == 1) mid_area[i] = 0.5 + dxy;
        else mid_area[i] = 0.5 - dxy;
    }
    else /* 2 pixels are covered by the edge in the ith column. */
    {
        if (dxy > 0.0)
        {
            if (top_bottom == 1)
            {
                up_area[i] = AA;
                mid_area[i] = T2 - up_area[i];
            }
            else
            {
                up_area[i] = 1.0 - AA;
                mid_area[i] = T1 - up_area[i];
            }
        }
        else
        {
            if (top_bottom == 1)
            {
                low_area[i] = 1.0 - AA;
                mid_area[i] = T1 - low_area[i];
            }
            else
            {
                low_area[i] = AA;
                mid_area[i] = T2 - low_area[i];
            }
        }
    }
    i = i + 1;
    if (i <= nm2) goto loop;

```

Figure 4-6. The algorithm DISLOP_E calculates the pixel areas on both sides divided by a passing polygon edge with a slope in the first range, [0,1]. See Figure 4-5.

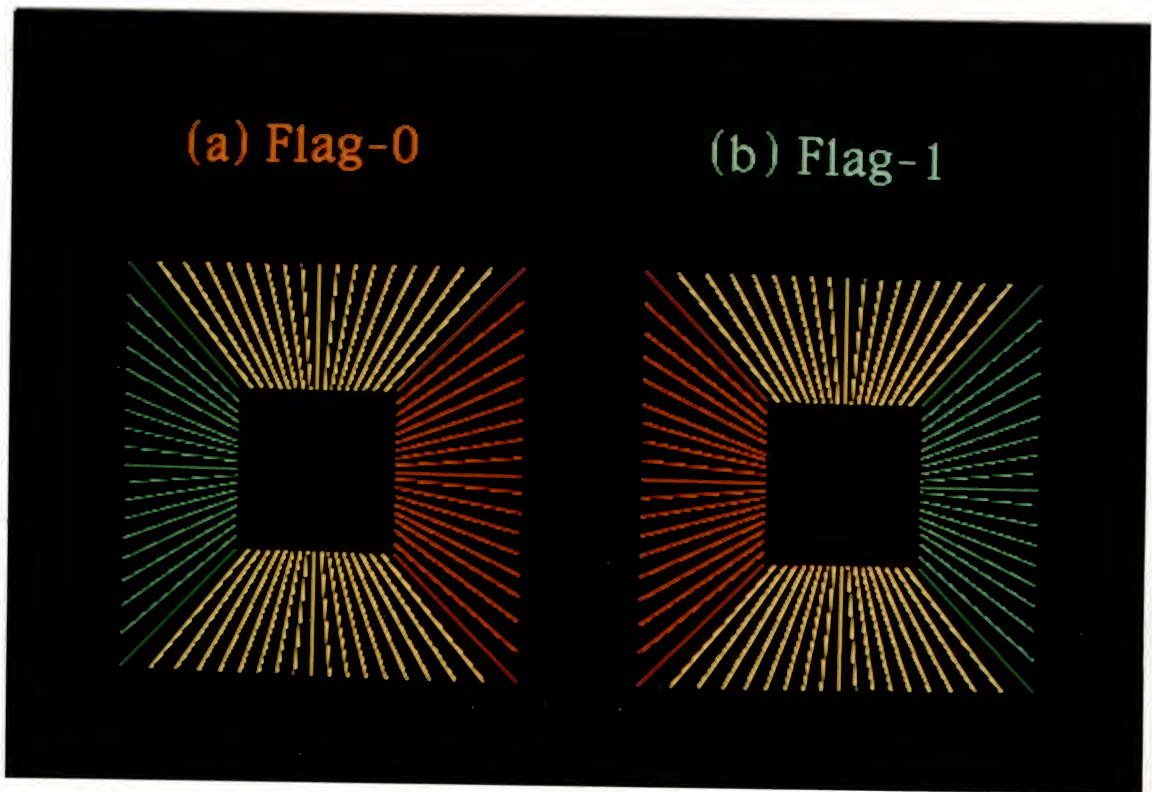


Figure 4-7. The flag assignments for polygon edges with slopes in the range of all four types.

color filled or shading polygons. Figure 4-8 shows the procedure of superimposing. A Chinese character is drawn as outlines of smoothed edges on the left and displayed as a set of different color filled polygons with jagged edges in the middle, and then by superimposing the left character on the middle one, a smoothed character is resulted in the right. Hence the edge smoothness of the two solid images with and without anti-aliasing can be apparently distinguished. However, some weird color spots appear on some middle parts of edges of triangles. This phenomenon reveals that the proper anti-aliasing can not be done by just simple superimposing. The background color has to be considered whenever it is not black. Solutions are discussed in the section of post-anti-aliasing.

Algorithm DISLOP C for Circle Rendering

Any curve can be represented by fitting it with a polygonal line. From experimental results, Fujimoto and Iwata [Fuji83] found that 0.2 pixels as the maximum distance between an arc and its inscribed polygonal line yields visually smooth arcs by polygonal lines. The number of sides of the polygon, n , is $\pi / \cos^{-1}(1 - T/R)$, where T is the tolerance and R is the radius of the arc. Since a circle is a constant radius arc, it can be represented by a polygonal line also. The above formula indicates how many sides the approximating polygon should have: to represent a circle with a radius of 17 pixels, a 20-sided inscribed polygon is needed according to the above formula [$n = \pi / \cos^{-1}(1 - 0.2/17) = 20$.] The accuracy is calculated as shown in Figure 4-9, i.e. 1.23% at most. This slightly worse than one bit in a byte, and may be perceptible in a high resolution display.

Since we will accurately calculate the partial pixel areas covered by edge elements of a circle, we need not follow the straight-line polygonal edge approximation indicated above. Bresenham's circle algorithm [Fole82] plots N pixels as the periphery of a circle of radius R , where $N = R / \sqrt{2} * 8$. Thus, 96 pixels are needed to display a circle with a 17-pixel radius. Each pixel may be viewed as an edge of an approximating polygon. Therefore, we

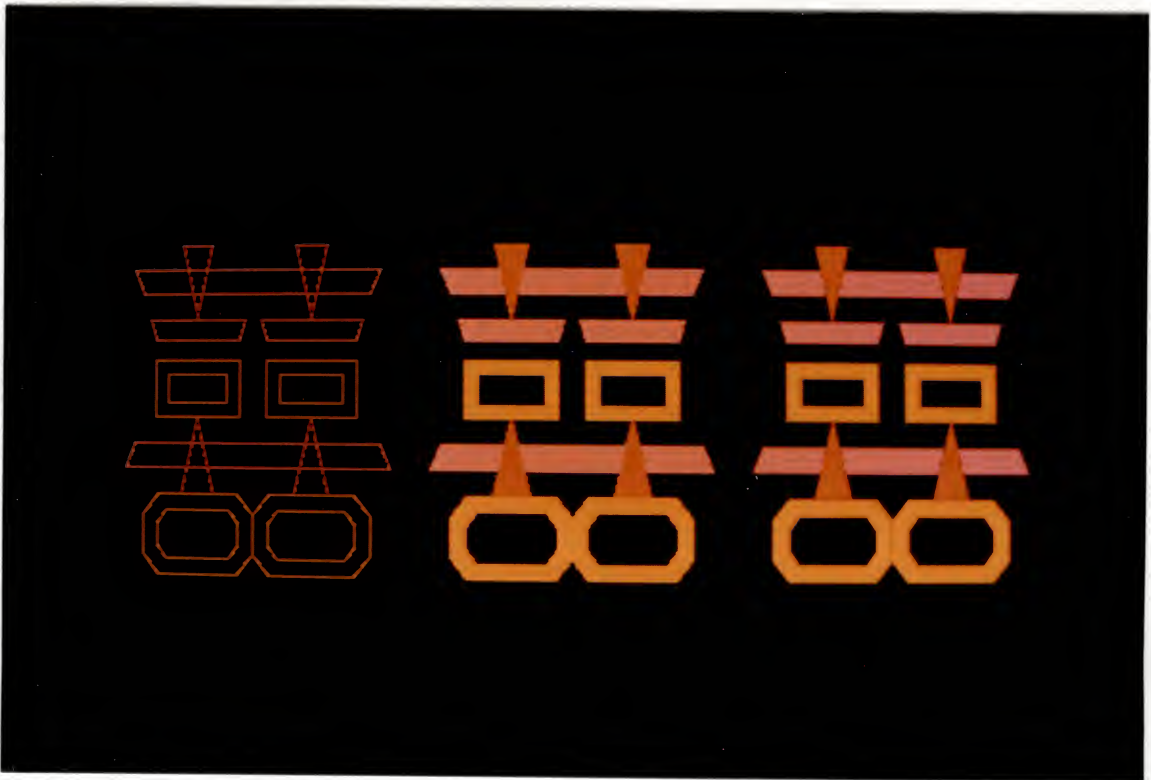
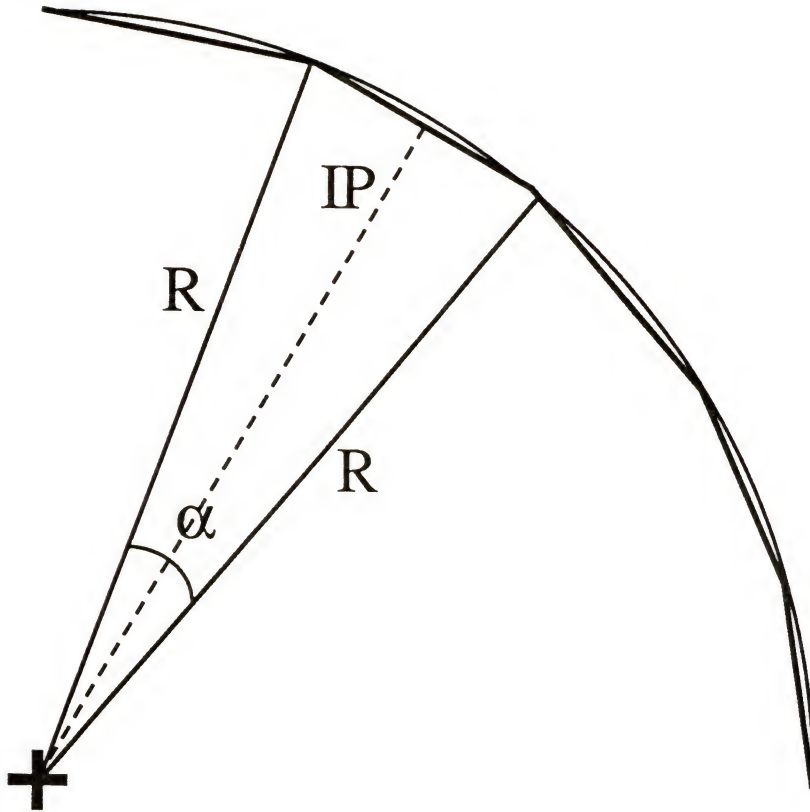


Figure 4-8. A Chinese character is displayed as outlines of smoothed edges (left), a set of different color filled polygons without anti-aliasing (middle) and with anti-aliasing but without the background color consideration (right).



$$\alpha = \frac{360}{20} = 18$$

$$\cos (0.5 * \alpha) = 0.9877$$

$$\frac{IP}{R} = 0.9877$$

$$\text{accuracy} = \frac{|IP - R|}{R} = 1 - 0.9877 = 0.0123 = 1.23\%$$

Figure 4-9. The calculation of accuracy about a circle of 17-pixel radius to a 20-sided inscribed polygon.

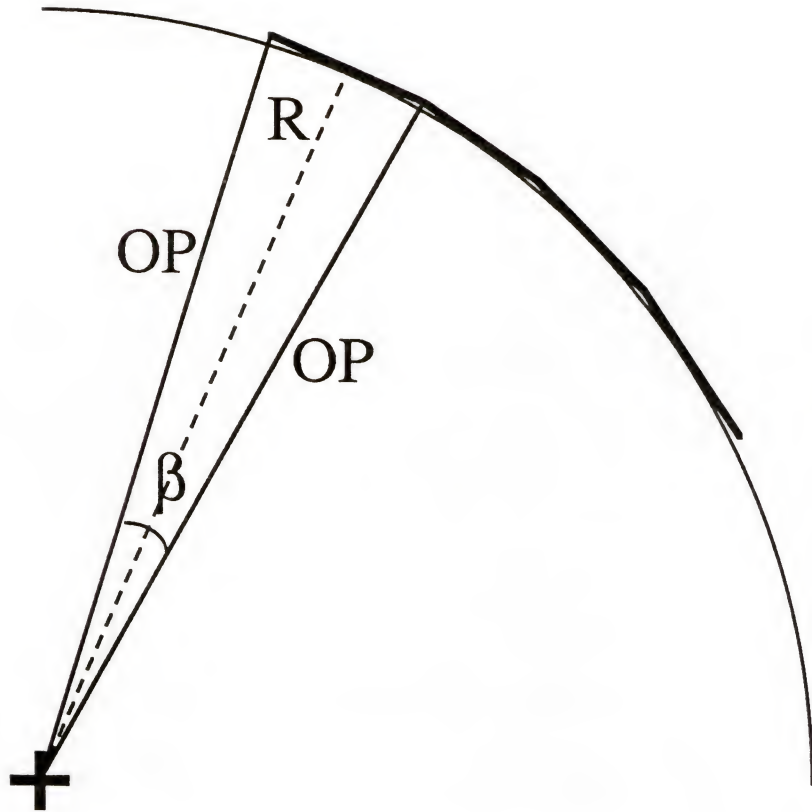
can draw a 96-sided polygon surrounding the circle of 17-pixel radius to represent it and obtain a much closer approximation, 0.05%. The calculation is shown in Figure 4-10.

Therefore, the method described here is precise enough for drawing polygon-fitted circles and also for shading the silhouette of spheres which is described in the next section. A polygon-fitted circle is actually a set of very short line segments. In other words, for rendering a circle of one-pixel thickness, the algorithm DISLOP_C can take advantage of the anti-aliasing line algorithm DISLOP_L for each line segment of the polygonal line representing the circle. Real values for the center and the radius of a circle are utilized in the algorithm DISLOP_C so it can anti-alias the circle properly. The results of a set of anti-aliased circles in different sizes from the algorithm DISLOP_C are shown in the right of Figure 4-11, whereas their corresponding non-anti-aliased circles from Bresenham's circle algorithm are on the left.

Algorithm DISLOP_S for Sphere Rendering

Spherical images displayed on raster devices usually show some aliasing effects on their silhouettes [Khur84]. An anti-aliasing approach developed by Jen [Jen84] is based on a weighting scheme that averages the display color intensities at the intersection of the actual sphere silhouette boundary with the edge of the area covered by the pixel. The procedure is approximate and Jen's error analysis of his algorithm shows that the maximum error is about 5% of a scan line for the smallest sphere being considered, i.e. the sphere with a radius of 5 pixels.

Since a circle can be represented by a polygonal line with high accuracy and visual smoothness, a sphere should as well be satisfactorily represented by a fitted polygon. Therefore, the model of polygon-fitted sphere is adopted as using a N-sided, $N = R / \sqrt{2} * 8$, polygon to fit a sphere with radius R. The radius does not have to be an integer. An example of a 26-sided polygon-fitted sphere with a center at (100.15,99.62)



$$\beta = \frac{360}{96} = 3.75$$

$$\cos (0.5 * \beta) = 0.99946$$

$$\frac{R}{OP} = 0.99946$$

$$\text{accuracy} = \frac{|OP - R|}{R} = \frac{\frac{R}{0.99946} - R}{R} = 0.000536 = 0.05\%$$

Figure 4-10. The calculation of accuracy about a circle of 17-pixel radius to a 96-sided polygon surrounding the circle.



Figure 4-11. Anti-aliased circles with varying radii using the algorithm DISLOP_C (right) and non-anti-aliased circles (left).

and a radius of 4.75 pixels is shown in Figure 4-12 and its error analysis is described as follows:

First, calculate values of two angles α and α_{11} :

$$\alpha = \sin^{-1} \left(\frac{dx}{R} \right) = \sin^{-1} \left(\frac{0.35}{4.75} \right) = 0.073751$$

$$\alpha_{11} = \sin^{-1} \left(\frac{dx_1}{R} \right) = \sin^{-1} \left(\frac{1.35}{4.75} \right) = 0.288183.$$

Then from their difference, we can get the value of the angle α_1 :

$$\alpha_1 = \alpha_{11} - \alpha = 0.288183 - 0.073751 = 0.214432$$

which is, in turn, used to calculate several areas:

$$ABC_Triangle_area1 = 0.5 * R * (R * \sin(\alpha_1)) = 2.400563$$

$$ABDC_Pie_area1 = \pi * R^2 * \frac{\alpha_1}{2 * \pi} = 2.419059$$

$$BDC_Chord_area1 = ABDC_Pie_area1 - ABC_Triangle_area1 = 0.018496.$$

Let the coordinate system be translated to (100.15, 99.62) such that the center of the sphere becomes the origin (0, 0). In this way, geometric calculations can be done much easier.

$$x_1 = dx + 0.5 = 0.35 + 0.5 = 0.85$$

$$y_1 = \sqrt{R^2 - x_1^2} = 4.673329$$

$$m_1 = \text{slope}_1 = -\frac{x_1}{y_1} = -\frac{0.85}{4.673329} = -0.181883$$

$$x_{1f} = x_{01} = 0.35$$

$$y_{1f} = y_1 + (x_{1f} - x_1) * m_1 = 4.764271$$

$$y_{01} = \sqrt{R^2 - x_{01}^2} = 4.737088$$

$$x_{1b} = x_{12} = 1.35$$

$$y_{1b} = y_1 + (x_{1b} - x_1) * m_1 = 4.582388$$

$$y_{12} = \sqrt{R^2 - x_{12}^2} = 4.554119$$

$$y_{1L} = y_{1R} = 3.88.$$

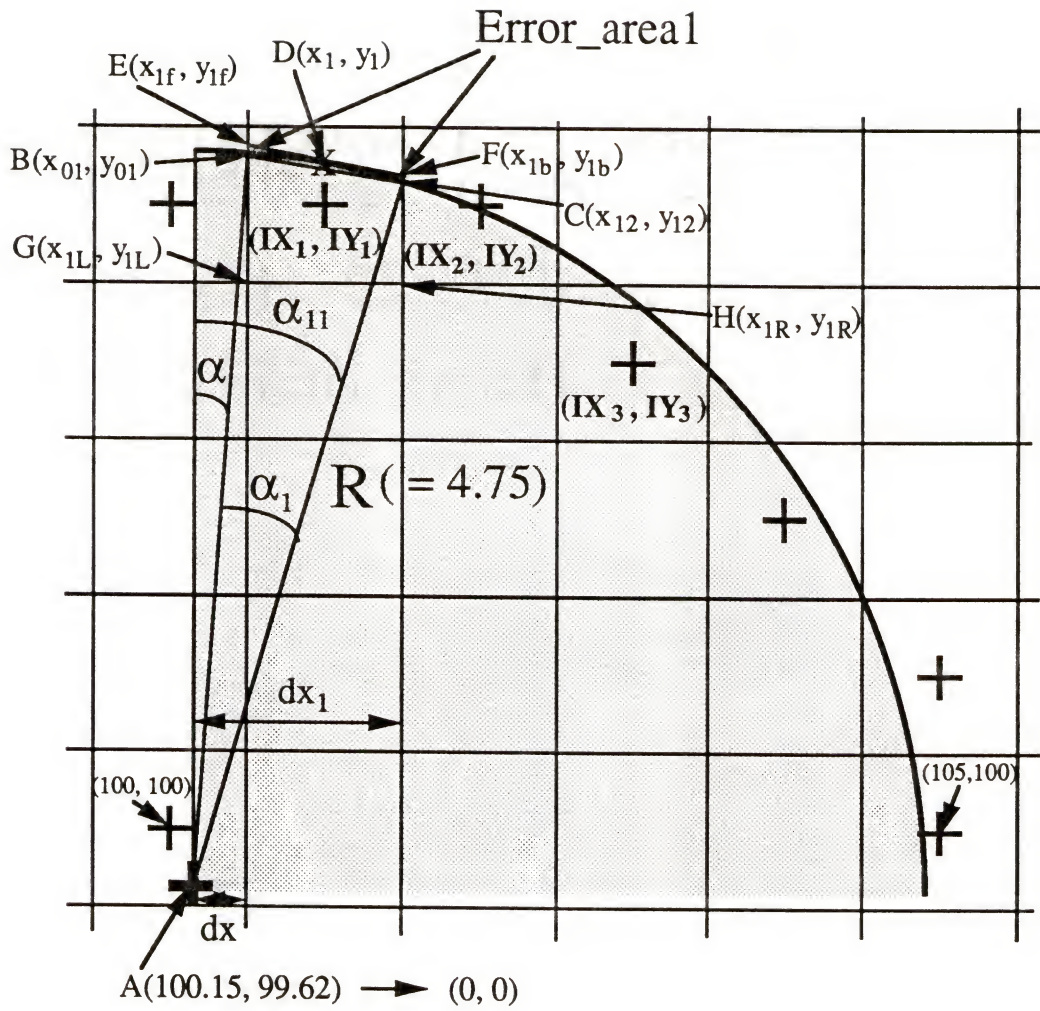


Figure 4-12. A polygon-fitted sphere with floating point values of center and radius.

In the pixel (IX_1, IY_1) , the algorithm DISLOP_S calculates the area of the trapezoid EFHG and regards it as the pixel coverage by the silhouette of the sphere for anti-aliasing:

$$EFHG_Trapezoid_area1 = \frac{(y_{1f} - y_{1L}) + (y_{1b} - y_{1R})}{2} = 0.793330.$$

However, the real covered area of the pixel is shaped like an arch, BDCHG, whose value is calculated as follows:

$$\begin{aligned} BDCHG_Arch_area1 &= BCHG_Trapezoid_area1 + BDC_Chord_area1 \\ &= \frac{(y_{01} - y_{1L}) + (y_{12} - y_{1R})}{2} + 0.018496 \\ &= 0.765604 + 0.018496 = 0.784100. \end{aligned}$$

Therefore, the error area is

$$Error_area1 = EFHG_Trapezoid_area1 - BDCHG_Arch_area1 = 0.009230$$

and the percentage of error on this pixel is

$$Error1 = \frac{Error_area1}{BDCHG_Arch_area1} = 0.011772 = 1.1772\%.$$

The error is about 1.2% for pixels on the silhouette of the sphere. For any polygon-fitted sphere with a larger radius, the angle α_1 is smaller indicating the curvature of the silhouette of the sphere in one pixel is smaller and the silhouette in any pixel can be more like a straight edge, which in turn determines the percentage of error being lesser than the above calculated value. Therefore, this model is highly acceptable.

The polygon-fitted sphere is actually a polygon with as many edges as the number of pixels required to draw the silhouette of the sphere. It is apparent that all edges of the polygon are very short, about one-pixel long. Therefore, to eliminate aliasing effects on the silhouettes of spherical images, the algorithm DISLOP_S utilizes the approach of anti-aliasing polygon edges, DISLOP_E, for each edge of the polygon representing the sphere. Floating point values for the center and the radius of sphere are also employed in the algorithm DISLOP_S for more proper anti-aliasing. The results of a set of anti-aliased

spheres as circular disks in different sizes from the algorithm DISLOP_S are shown in the right of Figure 4-13, whereas their corresponding non-anti-aliased circular disks are on the left.

Applications and Comparisons of Algorithms

The anti-aliasing approach of Fujimoto and Iwata [Fuji83] utilizes the special properties of the vector edge, namely its slope and thickness, which is geometrically equal to zero, and uses a spatial filter adopting two Fourier window sizes to shade 2 pixels and 3 pixels in each column, respectively, for a given vector slope. The algorithm states that the intensity of each pixel is a linear function of its distance from the center line of the vector and introduces an incremental calculation method to generate both the pixel's intensity and its position. This approach is very simple and efficient. However, it can be seen from the algorithm flowchart that the anti-aliased vectors produced by either shading 2 or 3 pixels in each column are not constant-intensity vectors, because the intensity per unit length is not constant. In addition the approach ignores the intensity of the upper pixel and/or the lower pixel, which may be as large as 32 for a line with 255 maximum intensity, whenever the center line of the vector passes through a pixel center and the vector's slope is not zero. Figure 4-14 shows how this algorithm works on smoothing vectors and results in non-constant-intensity vectors.

The algorithm DISLOP_L can generate constant-intensity, anti-aliased lines with end points of floating point coordinates and for any given slope. For a line one-pixel thick 1, 2 or 3 pixels are shaded in one column or one row according to the line's slope, m , and the vertical or horizontal distance, d_{xy} , between the pixel center and the center of the line in the same pixel square (see Figure 4-1). These two factors also determine the area of each pixel covered by the line. The total area, or intensity, of pixels covered by the line in each column (or row) is always constant. Figure 4-15 compares the pixels covered by anti-aliased lines using the algorithm DISLOP_L; Figure 4-14 shows those for Fujimoto's

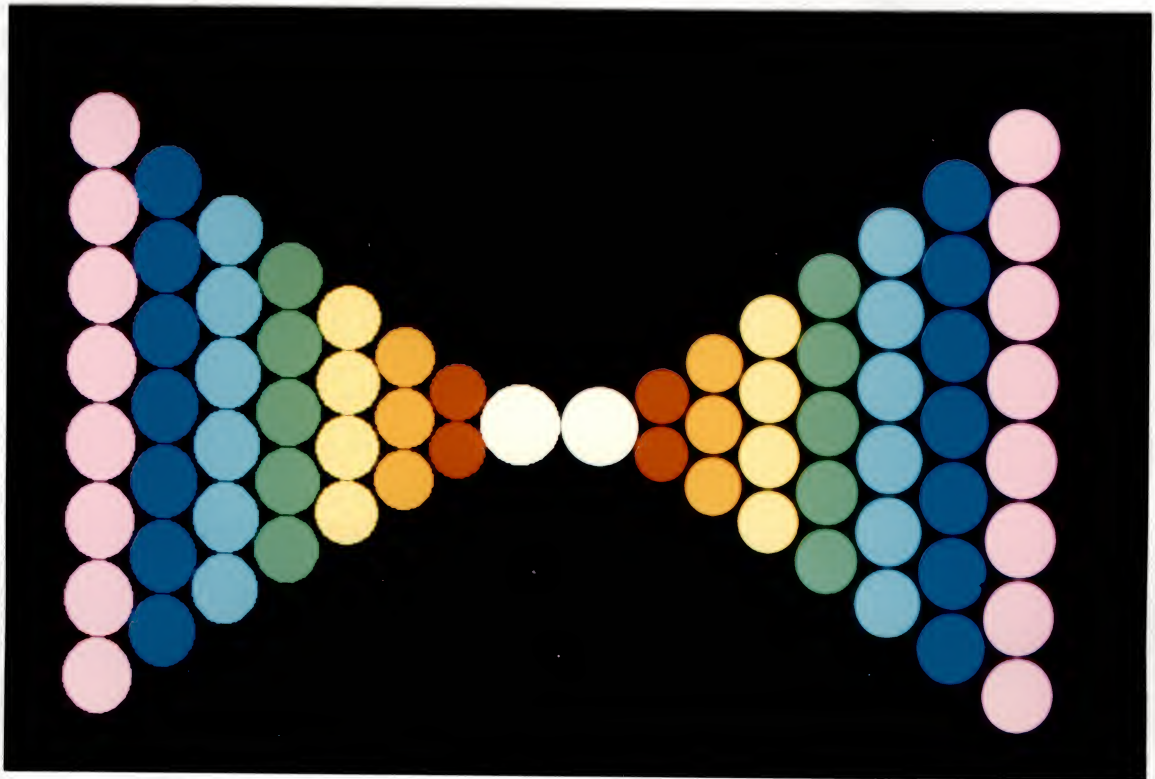


Figure 4-13. Anti-aliased spheres as circular disks (right) using the algorithm DISLOP_S and non-anti-aliased circular disks (left).

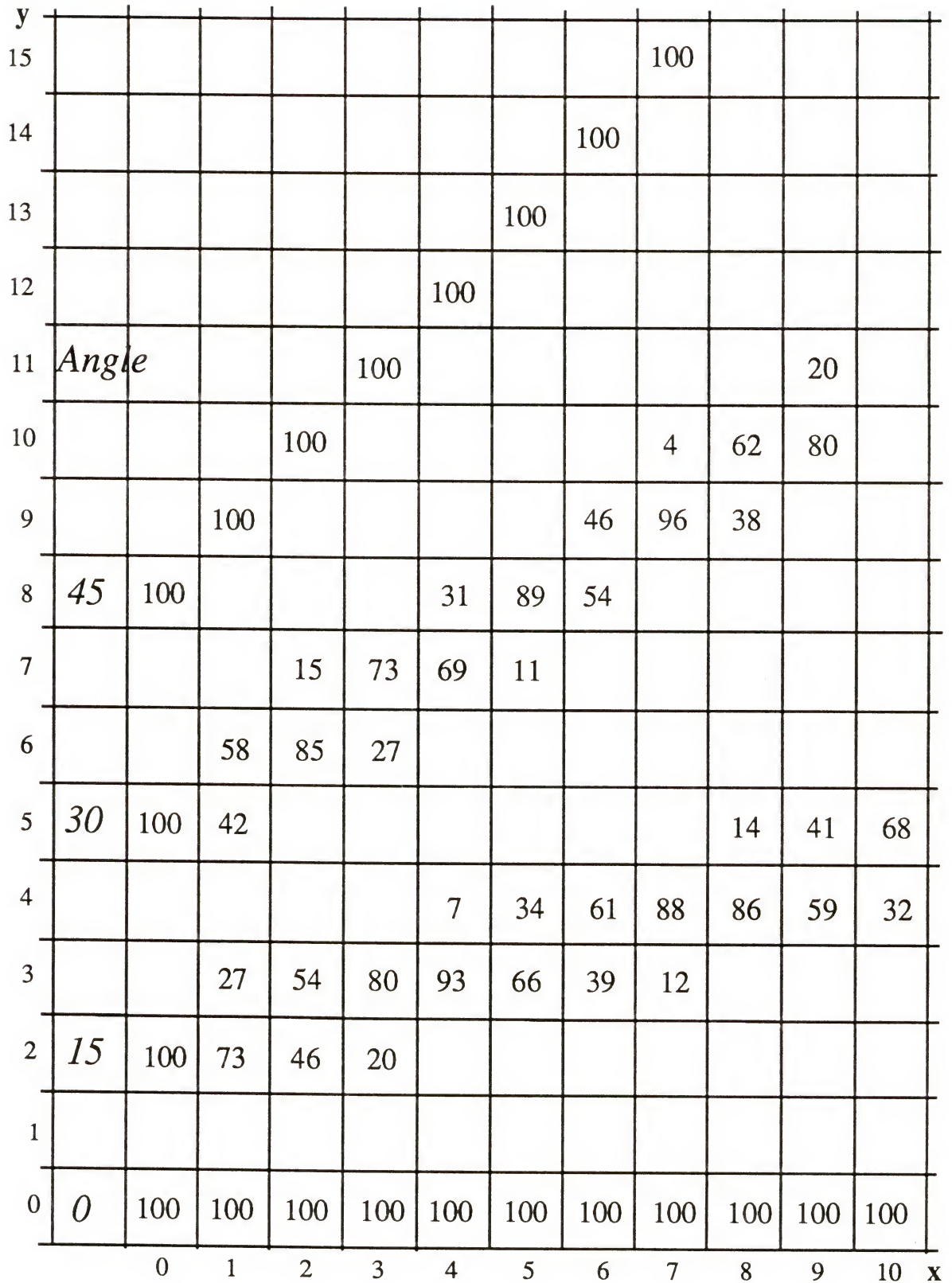


Figure 4-14. Anti-aliased lines of various slopes produced by the algorithm of Fujimoto and Iwata. Here, each line has an intensity 100 and is 10-pixel long. The number in each pixel indicates its intensity value in the corresponding line.

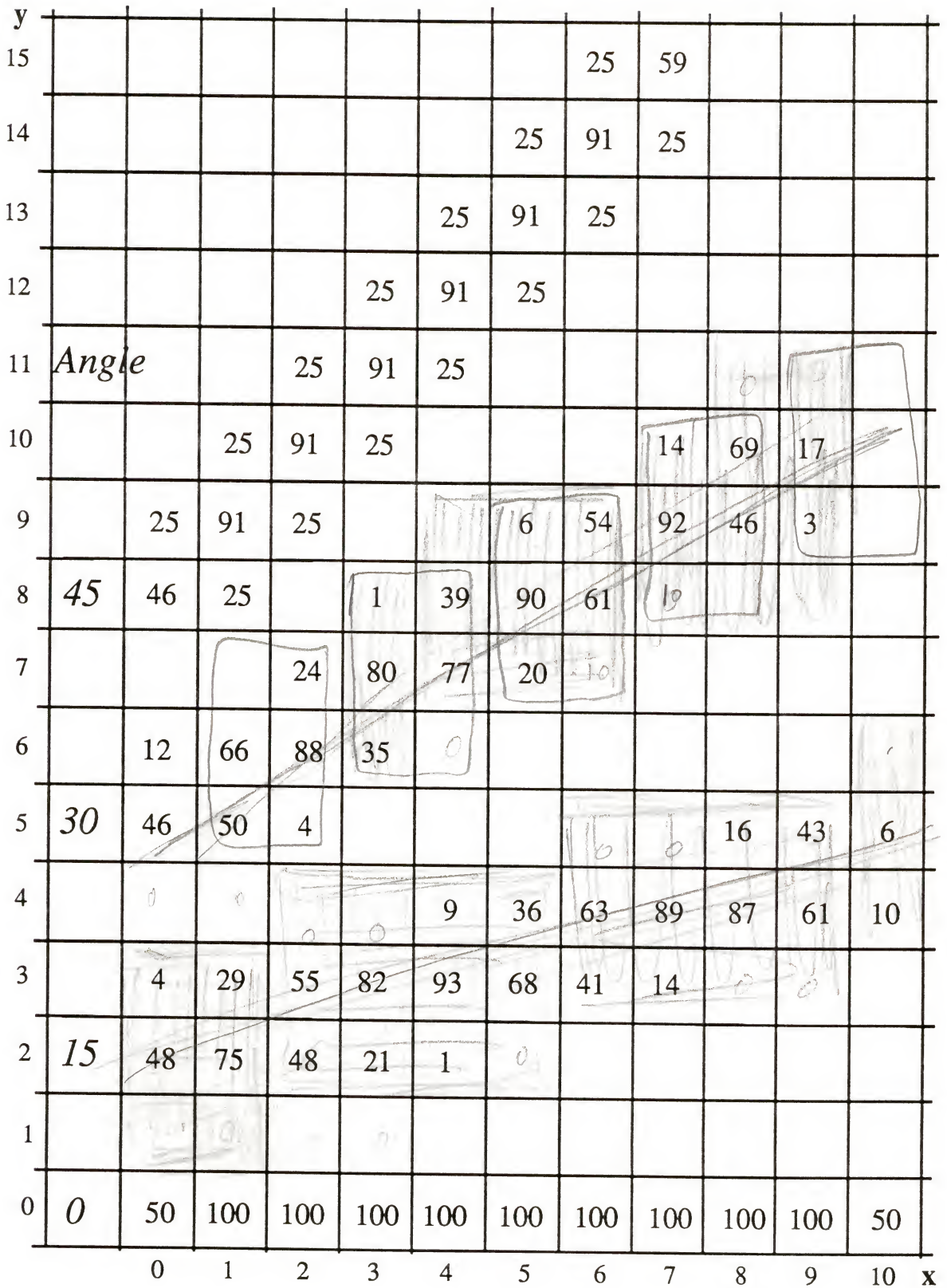


Figure 4-15. Anti-aliased lines of various slopes produced by the algorithm DISLOP_L. Here, each line has an intensity 100 and is 10-pixel long. The number in each pixel indicates its intensity value in the corresponding line.

algorithm. Obviously, the algorithm DISLOP_L renders lines as constant intensity, whereas the algorithm of Fujimoto and Iwata does not. Therefore, the anti-aliasing algorithm DISLOP_L is more accurate. The computation required is also very simple.

Devising simple algorithms for anti-aliasing functional smooth curves is of current research interest. One might try recursive supersampling with an idea of quadrees [Crow81a, Same84]. Here each pixel is subdivided into a regular grid of subpixels, which cut the pixel into a much finer regular grid. To determine the area coverage, the number of subpixels hit by the center of the line and its two edges are counted. For a given class of lines of interest one can use a large number of test cases to determine how many of the subpixels in fact are intersected. Not all possible lines are used, but rather only those that will occur for a given class. For example a line of 1 pixel width is found to hit at most 13 of the possible 16 subpixels for a smooth line rendered with a recursive supersampling algorithm.

Using a recursive algorithm one can render the functional curve directly rather than as a set of short straight line segments. It seemed peculiar that no pixel was covered by more than 13 subpixels; however, upon closer examination it is seen that the rendered line does not have the property of constant line intensity. This inconstancy of the line intensity leads to the "ropiness" of the line, an effect noted by Fujimoto and Iwata [Fuji83]. Also the rendering calculation time for $y = x^3 - 3 * x - 2$ (used as a test case) in the interval $x=[-2,2]$ was over one minute of VAX 11/780 CPU time. The DISLOP_L algorithm developed here may be used to render the curve as a set of very short line segments with various slopes. The result is a constant-intensity smoothed curve and the calculation time is less than 2 seconds for the same test case. The two cases, together with an aliased curve, are shown in Figure 4-16. It is noted that the anti-aliased curve of the DISLOP_L (the right one) is the smoothest of all. The pixel intensity values calculated in the two cases are also shown in Figure 4-17 for the same portion of the curve peak.

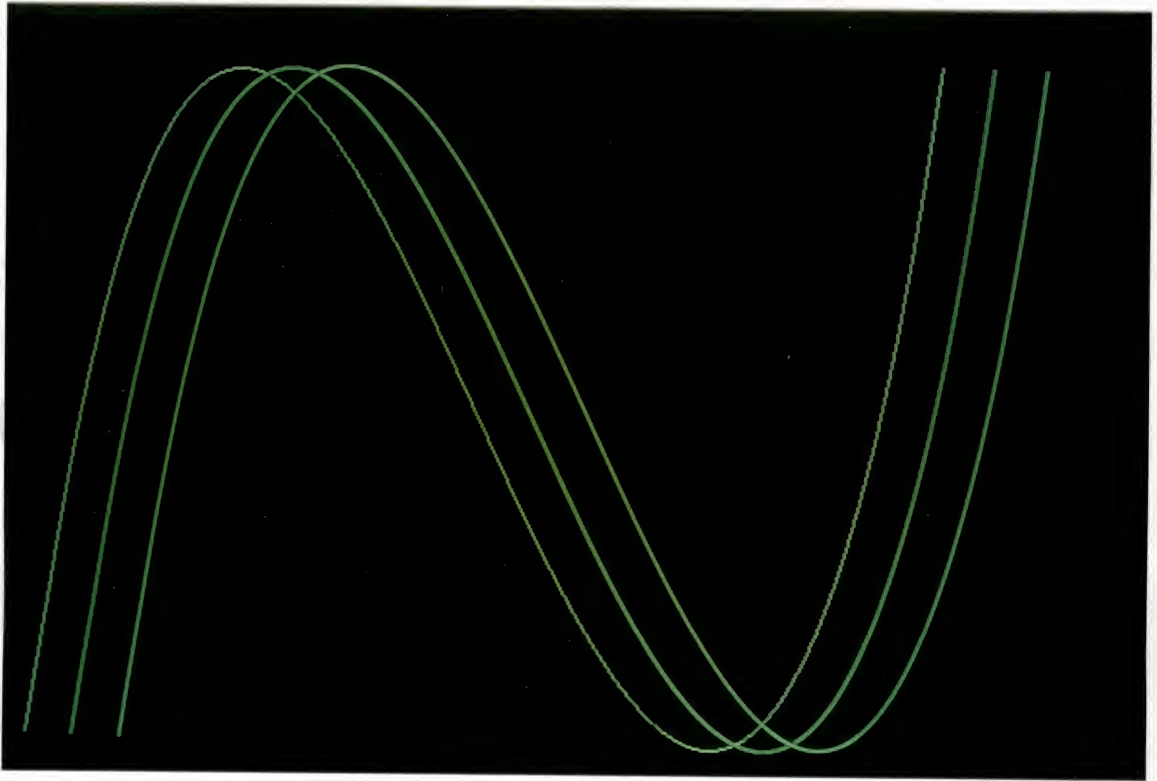


Figure 4-16. Comparison of a non-anti-aliased cubic curve (left), and anti-aliased curves using recursive supersampling (middle) and DISLOP_L (right) algorithms.

Generation of Area/Intensity Sampling Look-up Tables

The area sampling algorithms DISLOP_L and DISLOP_E eliminate aliasing effects on various kinds of object images shown on a raster device. In these algorithms two parameters, the slope m and the horizontal or vertical distance dxy , determine the accuracy of anti-aliasing. For more efficiently removing aliasing effects on images, generation of the area and intensity sampling look-up tables from these algorithms is required. This can be done by creating two-dimensional tables of area/intensity values for pixels covered by anti-aliased lines and polygon edges in accordance with the two parameters enumerated above. The development of the area/intensity sampling look-up tables involves the following procedures:

(1) Choices of the number of intensity levels are relevant to the size of the look-up tables and the visual smoothness of the anti-aliased images. Although 256 intensity levels, i.e. 8 bit-depth, are usually implemented for shading a realistic image, this number can be decreased for smoothing the silhouette of the image. According to experiences of some previous anti-aliasing methods, 16 equal intensity levels may be adequate for sampling jagged edges of images and thus will be tested with area/intensity sampling look-up tables as well as with 256 intensity levels.

(2) The sampling look-up tables in DISLOP_L and DISLOP_E will determine how much of a pixel is covered for a line (or edge) of a given slope. Such tables will be periodic: i.e. the slope will determine sections of the line that are repeating elements. The accuracy of the area coverage and the slope will determine the size of this periodicity. The period determines the size of the required storage areas for these tables.

The pixel areas covered by anti-aliased lines with a slope of 5° , or 0.0875, vary as the vertical distance dxy changes in each column as shown in rows in Table 4-1, in which the pixels' intensity values are calculated based on the chosen number of intensity levels, 16 or 256. The distance dxy in each column is the sum of that value in the previous column and the slope m , and is always in the range $[-0.5, 0.5]$. Values of dxy fluctuate within the pixel

Table 4-1. An anti-aliased line with a slope of 5° , or 0.0875, generated by the algorithm DISLOP_L. Areas and intensity values of pixels covered by the line are calculated based on the slope m, the distance dxy, and the chosen number of anti-aliasing intensity levels, 16 or 256.

Slope m	Distance dxy	Covered Pixel Areas			16 Intensity Levels			256 Intensity Levels		
		Upper	Middle	Lower	Up	Mid	Low	Up	Mid	Low
0.0875	0.0000	0.0119	0.9800	0.0119	0	15	0	3	250	3
0.0875	0.0875	0.0894	0.9144	0.0000	1	14	0	23	233	0
0.0875	0.1750	0.1769	0.8269	0.0000	3	12	0	45	211	0
0.0875	0.2625	0.2644	0.7394	0.0000	4	11	0	67	189	0
0.0875	0.3500	0.3519	0.6520	0.0000	5	10	0	90	166	0
0.0875	0.4374	0.4393	0.5645	0.0000	7	8	0	112	144	0
0.0875	-0.4751	0.0000	0.5268	0.4770	0	8	7	0	134	122
0.0875	-0.3876	0.0000	0.6143	0.3895	0	9	6	0	157	99
0.0875	-0.3001	0.0000	0.7018	0.3020	0	11	5	0	179	77
0.0875	-0.2126	0.0000	0.7893	0.2145	0	12	3	0	201	55
0.0875	-0.1251	0.0000	0.8768	0.1270	0	13	2	0	224	32
0.0875	-0.0376	0.0004	0.9638	0.0396	0	14	1	0	246	10
0.0875	0.0499	0.0518	0.9521	0.0000	1	14	0	13	243	0
0.0875	0.1373	0.1393	0.8646	0.0000	2	13	0	36	220	0
0.0875	0.2248	0.2267	0.7771	0.0000	3	12	0	58	198	0
0.0875	0.3123	0.3142	0.6896	0.0000	5	10	0	80	176	0
0.0875	0.3998	0.4017	0.6021	0.0000	6	9	0	102	154	0
0.0875	0.4873	0.4892	0.5146	0.0000	7	8	0	125	131	0
0.0875	-0.4252	0.0000	0.5767	0.4271	0	9	6	0	147	109
0.0875	-0.3377	0.0000	0.6642	0.3396	0	10	5	0	169	87
0.0875	-0.2502	0.0000	0.7517	0.2522	0	11	4	0	192	64
0.0875	-0.1628	0.0000	0.8392	0.1647	0	13	2	0	214	42
0.0875	-0.0753	0.0000	0.9266	0.0772	0	14	1	0	236	20
0.0875	0.0122	0.0191	0.9783	0.0064	0	15	0	5	249	2
0.0875	0.0997	0.1016	0.9022	0.0000	2	14	0	26	230	0
0.0875	0.1872	0.1891	0.8147	0.0000	3	12	0	48	208	0
0.0875	0.2747	0.2766	0.7272	0.0000	4	11	0	71	185	0
0.0875	0.3622	0.3641	0.6397	0.0000	5	10	0	93	163	0
0.0875	0.4497	0.4516	0.5523	0.0000	7	8	0	115	141	0
0.0875	-0.4629	0.0000	0.5391	0.4648	0	8	7	0	137	119
0.0875	-0.3754	0.0000	0.6265	0.3773	0	9	6	0	160	96
0.0875	-0.2879	0.0000	0.7140	0.2898	0	11	4	0	182	74
0.0875	-0.2004	0.0000	0.8015	0.2023	0	12	3	0	204	52
0.0875	-0.1129	0.0000	0.8890	0.1148	0	13	2	0	227	29
0.0875	-0.0254	0.0023	0.9726	0.0289	0	15	0	1	248	7
0.0875	0.0621	0.0640	0.9398	0.0000	1	14	0	16	240	0
0.0875	0.1496	0.1515	0.8524	0.0000	2	13	0	39	217	0
0.0875	0.2370	0.2390	0.7649	0.0000	4	11	0	61	195	0
0.0875	0.3245	0.3264	0.6774	0.0000	5	10	0	83	173	0
0.0875	0.4120	0.4139	0.5899	0.0000	6	9	0	106	150	0
0.0875	0.4995	0.5014	0.5024	0.0000	8	8	0	128	128	0
0.0875	-0.4130	0.0000	0.5889	0.4149	0	9	6	0	150	106
0.0875	-0.3255	0.0000	0.6764	0.3274	0	10	5	0	172	83
0.0875	-0.2380	0.0000	0.7639	0.2399	0	11	4	0	195	61
0.0875	-0.1505	0.0000	0.8514	0.1524	0	13	2	0	217	39
0.0875	-0.0631	0.0000	0.9389	0.0650	0	14	1	0	239	17
0.0875	0.0244	0.0281	0.9732	0.0026	0	15	0	7	248	1
0.0875	0.1119	0.1138	0.8900	0.0000	2	13	0	29	227	0
0.0875	0.1994	0.2013	0.8025	0.0000	3	12	0	51	205	0
0.0875	0.2869	0.2888	0.7150	0.0000	4	11	0	74	182	0
0.0875	0.3744	0.3763	0.6275	0.0000	6	9	0	96	160	0
0.0875	0.4619	0.4638	0.5400	0.0000	7	8	0	118	138	0

Table 4-1--continued.

Slope m	Distance dxy	Covered Pixel Areas			16 Intensity Levels			256 Intensity Levels		
		Upper	Middle	Lower	Up	Mid	Low	Up	Mid	Low
0.0875	-0.4506	0.0000	0.5513	0.4525	0	8	7	0	141	115
0.0875	-0.3631	0.0000	0.6388	0.3651	0	10	5	0	163	93
0.0875	-0.2757	0.0000	0.7262	0.2776	0	11	4	0	185	71
0.0875	-0.1882	0.0000	0.8137	0.1901	0	12	3	0	208	48
0.0875	-0.1007	0.0000	0.9012	0.1026	0	14	2	0	230	26
0.0875	-0.0132	0.0060	0.9780	0.0198	0	15	0	2	249	5
0.0875	0.0743	0.0762	0.9276	0.0000	1	14	0	19	237	0
0.0875	0.1618	0.1637	0.8401	0.0000	2	13	0	42	214	0
0.0875	0.2493	0.2512	0.7526	0.0000	4	11	0	64	192	0
0.0875	0.3368	0.3387	0.6652	0.0000	5	10	0	86	170	0
0.0875	0.4242	0.4262	0.5777	0.0000	6	9	0	109	147	0
0.0875	-0.4883	0.0000	0.5136	0.4902	0	8	7	0	131	125
0.0875	-0.4008	0.0000	0.6011	0.4027	0	9	6	0	153	103
0.0875	-0.3133	0.0000	0.6886	0.3152	0	10	5	0	176	80
0.0875	-0.2258	0.0000	0.7761	0.2277	0	12	3	0	198	58
0.0875	-0.1383	0.0000	0.8636	0.1402	0	13	2	0	220	36
0.0875	-0.0508	0.0000	0.9511	0.0527	0	14	1	0	243	13
0.0875	0.0367	0.0387	0.9646	0.0005	1	14	0	10	246	0
0.0875	0.1241	0.1261	0.8778	0.0000	2	13	0	32	224	0
0.0875	0.2116	0.2135	0.7903	0.0000	3	12	0	54	202	0
0.0875	0.2991	0.3010	0.7028	0.0000	5	11	0	77	179	0
0.0875	0.3866	0.3885	0.6153	0.0000	6	9	0	99	157	0
0.0875	0.4741	0.4760	0.5278	0.0000	7	8	0	121	135	0
0.0875	-0.4384	0.0000	0.5635	0.4403	0	8	7	0	144	112
0.0875	-0.3509	0.0000	0.6510	0.3528	0	10	5	0	166	90
0.0875	-0.2634	0.0000	0.7385	0.2654	0	11	4	0	188	68
0.0875	-0.1760	0.0000	0.8260	0.1779	0	12	3	0	211	45
0.0875	-0.0885	0.0000	0.9134	0.0904	0	14	1	0	233	23
0.0875	-0.0010	0.0114	0.9800	0.0124	0	15	0	3	250	3
0.0875	0.0865	0.0884	0.9154	0.0000	1	14	0	23	233	0
0.0875	0.1740	0.1759	0.8279	0.0000	3	12	0	45	211	0
0.0875	0.2615	0.2634	0.7404	0.0000	4	11	0	67	189	0
0.0875	0.3490	0.3509	0.6529	0.0000	5	10	0	89	166	0
0.0875	0.4365	0.4384	0.5655	0.0000	7	8	0	112	144	0
0.0875	-0.4761	0.0000	0.5259	0.4780	0	8	7	0	134	122
0.0875	-0.3886	0.0000	0.6133	0.3905	0	9	6	0	156	100
0.0875	-0.3011	0.0000	0.7008	0.3030	0	11	5	0	179	77
0.0875	-0.2136	0.0000	0.7883	0.2155	0	12	3	0	201	55
0.0875	-0.1261	0.0000	0.8758	0.1280	0	13	2	0	223	33
0.0875	-0.0386	0.0003	0.9630	0.0406	0	14	1	0	246	10
0.0875	0.0489	0.0508	0.9530	0.0000	1	14	0	13	243	0
0.0875	0.1364	0.1383	0.8655	0.0000	2	13	0	35	221	0
0.0875	0.2238	0.2258	0.7781	0.0000	3	12	0	58	198	0
0.0875	0.3113	0.3132	0.6906	0.0000	5	10	0	80	176	0
0.0875	0.3988	0.4007	0.6031	0.0000	6	9	0	102	154	0
0.0875	0.4863	0.4882	0.5156	0.0000	7	8	0	124	131	0
0.0875	-0.4262	0.0000	0.5757	0.4281	0	9	6	0	147	109
0.0875	-0.3387	0.0000	0.6632	0.3406	0	10	5	0	169	87

square of different columns; this fluctuation is almost periodic for the multiples of the increments in m being approximately an integer. For instance, multiplying the increment 0.08749... by 23, we get the product 2.0122... which is very close to the integer 2 (see Table 4-2); whereas changing the multiplier 23 to 80, 6.9990... is the product and very close to the integer 7 (see Table 4-3), too. Their accuracies are 0.0122 and 0.0010 for values of dxy in periods of 23 and 80 columns, respectively. This accuracy can be applied to the pixel area coverage. Its relationship to dxy can be seen from the algorithms DISLOP_L in Figure 4-2 and DISLOP_E in Figure 4-6. In other words, an anti-aliased line covers almost the same amount of pixel areas over a series of columns in each period. Since the real value of the area coverage in each pixel is multiplied by the chosen number of anti-aliasing intensity levels and is rounded to an integer value, the precision in intensity values of pixels can be quite acceptable by choosing the proper number of intensity levels for different periods. In Table 4-2 and 4-3, two intensity levels, 16 and 256, set two tolerances, 0.0667 and 0.0039, on the pixel area coverage, respectively, and thus result in different approximated-periodicities and precisions in areas and intensity values for pixels covered by a set of anti-aliased lines with slopes of 0° , 1° , 2° , ..., and 45° .

Many large numbers of periods in Table 4-3 show that generation of the area/intensity sampling look-up tables for 256 intensity levels could be impractical and unnecessary. On the other hand, 16 intensity levels for anti-aliasing images may not give a fine enough shading. An acceptable compromise may be 64 intensity levels as shown in Table 4-4. For anti-aliasing the line of 5° in Table 4-1 at 64 intensity levels, the period is now 57 columns instead of 23 or 80 columns, and its area/intensity sampling look-up tables can be generated as in Table 4-5. Here, less than 200 bytes is needed for storing the table containing integer intensity values which then may be used for quickly smoothing this particular line. Another example is Table 4-6, which shows the generation of the area/intensity sampling look-up tables for anti-aliasing an edge on the top of a polygon with a slope of 40° at 64 intensity levels for a period of 56 columns.

Table 4-2. The chosen number of anti-aliasing intensity levels is 16.

Vector's Degree	Slope (m)	Multiple (Period)	Value (m*Period)	Closest Integer	Precision in Area	Precision in Intensity
0	0.0000	1	0.0000	0	0.0000	0.0000
1	0.0175	55	0.9600	1	0.0400	0.6397
2	0.0349	29	1.0127	1	0.0127	0.2031
3	0.0524	19	0.9957	1	0.0043	0.0682
4	0.0699	43	3.0068	3	0.0068	0.1091
5	0.0875	23	2.0122	2	0.0122	0.1955
6	0.1051	19	1.9970	2	0.0030	0.0486
7	0.1228	49	6.0164	6	0.0164	0.2621
8	0.1405	50	7.0270	7	0.0270	0.4315
9	0.1584	19	3.0093	3	0.0093	0.1484
10	0.1763	17	2.9975	3	0.0025	0.0396
11	0.1944	36	6.9976	7	0.0024	0.0381
12	0.2126	33	7.0143	7	0.0143	0.2287
13	0.2309	13	3.0013	3	0.0013	0.0201
14	0.2493	52	12.9649	13	0.0351	0.5613
15	0.2679	41	10.9858	11	0.0142	0.2272
16	0.2867	56	16.0576	16	0.0576	0.9211
17	0.3057	36	11.0062	11	0.0062	0.0989
18	0.3249	37	12.0219	12	0.0219	0.3503
19	0.3443	29	9.9854	10	0.0146	0.2338
20	0.3640	11	4.0036	4	0.0036	0.0580
21	0.3839	52	19.9607	20	0.0393	0.6288
22	0.4040	47	18.9890	19	0.0110	0.1758
23	0.4245	33	14.0075	14	0.0075	0.1201
24	0.4452	54	24.0421	24	0.0421	0.6731
25	0.4663	15	6.9945	7	0.0055	0.0875
26	0.4877	39	19.0213	19	0.0213	0.3415
27	0.5095	51	25.9855	26	0.0145	0.2322
28	0.5317	32	17.0145	17	0.0145	0.2319
29	0.5543	56	31.0409	31	0.0409	0.6548
30	0.5773	26	15.0109	15	0.0109	0.1747
31	0.6009	50	30.0426	30	0.0426	0.6824
32	0.6249	8	4.9989	5	0.0011	0.0178
33	0.6494	20	12.9880	13	0.0120	0.1923
34	0.6745	40	26.9800	27	0.0200	0.3203
35	0.7002	10	7.0020	7	0.0020	0.0317
36	0.7265	51	37.0532	37	0.0532	0.8505
37	0.7535	57	42.9520	43	0.0480	0.7682
38	0.7813	32	25.0008	25	0.0008	0.0126
39	0.8098	21	17.0052	17	0.0052	0.0835
40	0.8391	31	26.0117	26	0.0117	0.1873
41	0.8693	23	19.9933	20	0.0067	0.1072
42	0.9004	10	9.0039	9	0.0039	0.0624
43	0.9325	59	55.0175	55	0.0175	0.2805
44	0.9657	29	28.0045	28	0.0045	0.0724
45	1.0000	1	1.0000	1	0.0000	0.0003

Table 4-3. The chosen number of anti-aliasing intensity levels is 256.

Vector's Degree	Slope (m)	Multiple (Period)	Value (m*Period)	Closest Integer	Precision in Area	Precision in Intensity
0	0.0000	1	0.0000	0	0.0000	0.0000
1	0.0175	172	3.0022	3	0.0022	0.5735
2	0.0349	86	3.0032	3	0.0032	0.8077
3	0.0524	229	12.0013	12	0.0013	0.3218
4	0.0699	143	9.9994	10	0.0006	0.1460
5	0.0875	80	6.9990	7	0.0010	0.2509
6	0.1051	314	33.0024	33	0.0024	0.6104
7	0.1228	57	6.9986	7	0.0014	0.3464
8	0.1405	185	25.9998	26	0.0002	0.0557
9	0.1584	101	15.9967	16	0.0033	0.8555
10	0.1763	380	67.0035	67	0.0035	0.9062
11	0.1944	391	76.0019	76	0.0019	0.4844
12	0.2126	207	43.9987	44	0.0013	0.3232
13	0.2309	758	174.9962	175	0.0038	0.9727
14	0.2493	365	91.0037	91	0.0037	0.9570
15	0.2679	209	56.0008	56	0.0008	0.1992
16	0.2867	136	38.9969	39	0.0031	0.7822
17	0.3057	157	47.9992	48	0.0008	0.2080
18	0.3249	277	90.0018	90	0.0018	0.4492
19	0.3443	61	21.0038	21	0.0038	0.9604
20	0.3640	272	98.9988	99	0.0012	0.3105
21	0.3839	99	38.0021	38	0.0021	0.5391
22	0.4040	99	39.9981	40	0.0019	0.4766
23	0.4245	139	59.0013	59	0.0013	0.3379
24	0.4452	146	65.0026	65	0.0026	0.6719
25	0.4663	178	83.0018	83	0.0018	0.4570
26	0.4877	285	139.0021	139	0.0021	0.5430
27	0.5095	210	106.9991	107	0.0009	0.2402
28	0.5317	205	108.9991	109	0.0009	0.2305
29	0.5543	175	97.0029	97	0.0029	0.7383
30	0.5773	97	56.0023	56	0.0023	0.5830
31	0.6009	233	139.9987	140	0.0013	0.3203
32	0.6249	877	548.0034	548	0.0034	0.8594
33	0.6494	77	50.0037	50	0.0037	0.9561
34	0.6745	255	171.9974	172	0.0026	0.6680
35	0.7002	487	340.9965	341	0.0035	0.8984
36	0.7265	128	92.9962	93	0.0038	0.9824
37	0.7535	142	107.0032	107	0.0032	0.8145
38	0.7813	32	25.0008	25	0.0008	0.2012
39	0.8098	184	148.9981	149	0.0019	0.4844
40	0.8391	87	73.0006	73	0.0006	0.1523
41	0.8693	153	132.9989	133	0.0011	0.2852
42	0.9004	251	225.9980	226	0.0020	0.5234
43	0.9325	163	151.9976	152	0.0024	0.6172
44	0.9657	204	196.9974	197	0.0026	0.6758
45	1.0000	1	1.0000	1	0.0000	0.0042

Table 4-4. The chosen number of anti-aliasing intensity levels is 64.

Vector's Degree	Slope (m)	Multiple (Period)	Value (m*Period)	Closest Integer	Precision in Area	Precision in Intensity
0	0.0000	1	0.0000	0	0.0000	0.0000
1	0.0175	57	0.9949	1	0.0051	0.3246
2	0.0349	57	1.9905	2	0.0095	0.6104
3	0.0524	57	2.9872	3	0.0128	0.8184
4	0.0699	57	3.9858	4	0.0142	0.9097
5	0.0875	57	4.9868	5	0.0132	0.8447
6	0.1051	57	5.9909	6	0.0091	0.5838
7	0.1228	57	6.9986	7	0.0014	0.0866
8	0.1405	57	8.0107	8	0.0107	0.6876
9	0.1584	82	12.9874	13	0.0126	0.8073
10	0.1763	17	2.9975	3	0.0025	0.1583
11	0.1944	36	6.9976	7	0.0024	0.1525
12	0.2126	80	17.0043	17	0.0043	0.2780
13	0.2309	13	3.0013	3	0.0013	0.0803
14	0.2493	349	87.0145	87	0.0145	0.9302
15	0.2679	56	15.0050	15	0.0050	0.3196
16	0.2867	129	36.9897	37	0.0103	0.6560
17	0.3057	72	22.0124	22	0.0124	0.7914
18	0.3249	40	12.9966	13	0.0034	0.2148
19	0.3443	61	21.0038	21	0.0038	0.2401
20	0.3640	228	82.9843	83	0.0157	1.0063
21	0.3839	86	33.0119	33	0.0119	0.7637
22	0.4040	52	21.0091	21	0.0091	0.5839
23	0.4245	66	28.0150	28	0.0150	0.9609
24	0.4452	128	56.9886	57	0.0114	0.7292
25	0.4663	148	69.0127	69	0.0127	0.8140
26	0.4877	41	19.9968	20	0.0032	0.2050
27	0.5095	51	25.9855	26	0.0145	0.9288
28	0.5317	79	42.0045	42	0.0045	0.2900
29	0.5543	83	46.0071	46	0.0071	0.4534
30	0.5773	71	40.9914	41	0.0086	0.5532
31	0.6009	218	130.9859	131	0.0141	0.8994
32	0.6249	8	4.9989	5	0.0011	0.0710
33	0.6494	57	37.0158	37	0.0158	1.0081
34	0.6745	43	29.0035	29	0.0035	0.2228
35	0.7002	50	35.0099	35	0.0099	0.6340
36	0.7265	106	77.0124	77	0.0124	0.7964
37	0.7535	69	51.9945	52	0.0055	0.3516
38	0.7813	32	25.0008	25	0.0008	0.0503
39	0.8098	63	51.0157	51	0.0157	1.0020
40	0.8391	56	46.9889	47	0.0111	0.7112
41	0.8693	107	93.0123	93	0.0123	0.7866
42	0.9004	221	198.9863	199	0.0137	0.8799
43	0.9325	74	69.0050	69	0.0050	0.3228
44	0.9657	29	28.0045	28	0.0045	0.2897
45	1.0000	1	1.0000	1	0.0000	0.0010

Table 4-5. The area/intensity sampling look-up tables for anti-aliasing a line with a slope of 5° , i.e. 0.0875, at 64 intensity levels.

Distance dxy => j	Pixel Area Coverage			64 Intensity Levels		
	Upper	Middle	Lower	Upper	Middle	Lower
-0.5000	0	0.0000	0.5019	0	32	32
-0.4825	1	0.0000	0.5195	0	33	31
-0.4649	2	0.0000	0.5370	0	34	29
-0.4474	3	0.0000	0.5545	0	35	28
-0.4298	4	0.0000	0.5721	0	36	27
-0.4123	5	0.0000	0.5896	0	37	26
-0.3947	6	0.0000	0.6072	0	38	25
-0.3772	7	0.0000	0.6247	0	39	24
-0.3596	8	0.0000	0.6423	0	40	23
-0.3421	9	0.0000	0.6598	0	42	22
-0.3246	10	0.0000	0.6773	0	43	21
-0.3070	11	0.0000	0.6949	0	44	19
-0.2895	12	0.0000	0.7124	0	45	18
-0.2719	13	0.0000	0.7300	0	46	17
-0.2544	14	0.0000	0.7475	0	47	16
-0.2368	15	0.0000	0.7651	0	48	15
-0.2193	16	0.0000	0.7826	0	49	14
-0.2018	17	0.0000	0.8002	0	50	13
-0.1842	18	0.0000	0.8177	0	52	12
-0.1667	19	0.0000	0.8352	0	53	11
-0.1491	20	0.0000	0.8528	0	54	10
-0.1316	21	0.0000	0.8703	0	55	8
-0.1140	22	0.0000	0.8879	0	56	7
-0.0965	23	0.0000	0.9054	0	57	6
-0.0789	24	0.0000	0.9230	0	58	5
-0.0614	25	0.0000	0.9405	0	59	4
-0.0439	26	0.0000	0.9580	0	60	3
-0.0263	27	0.0021	0.9721	0	61	2
-0.0088	28	0.0078	0.9791	0	62	1
0.0088	29	0.0169	0.9791	1	62	0
0.0263	30	0.0296	0.9721	2	61	0
0.0439	31	0.0458	0.9580	3	60	0
0.0614	32	0.0633	0.9405	4	59	0
0.0789	33	0.0809	0.9230	5	58	0
0.0965	34	0.0984	0.9054	6	57	0
0.1140	35	0.1159	0.8879	7	56	0
0.1316	36	0.1335	0.8703	8	55	0
0.1491	37	0.1510	0.8528	10	54	0
0.1667	38	0.1686	0.8352	11	53	0
0.1842	39	0.1861	0.8177	12	52	0
0.2018	40	0.2037	0.8002	13	50	0
0.2193	41	0.2212	0.7826	14	49	0
0.2368	42	0.2388	0.7651	15	48	0
0.2544	43	0.2563	0.7475	16	47	0
0.2719	44	0.2738	0.7300	17	46	0
0.2895	45	0.2914	0.7124	18	45	0
0.3070	46	0.3089	0.6949	19	44	0
0.3246	47	0.3265	0.6773	21	43	0
0.3421	48	0.3440	0.6598	22	42	0
0.3596	49	0.3616	0.6423	23	40	0
0.3772	50	0.3791	0.6247	24	39	0
0.3947	51	0.3966	0.6072	25	38	0
0.4123	52	0.4142	0.5896	26	37	0
0.4298	53	0.4317	0.5721	27	36	0
0.4474	54	0.4493	0.5545	28	35	0
0.4649	55	0.4668	0.5370	29	34	0
0.4825	56	0.4844	0.5195	31	33	0
0.5000	57	0.5019	0.5019	32	32	0

Table 4-6. The area/intensity sampling look-up tables for smoothing an edge on the top of a polygon with a slope of 40° , i.e. 0.8391, at 64 intensity levels.

Distance dxy => j	j	Pixel Area Coverage			64 Intensity Levels		
		Upper	Middle	Lower	Upper	Middle	Lower
-0.5000	0	0.0000	0.1049	0.8951	0	7	56
-0.4821	1	0.0000	0.1140	0.9039	0	7	57
-0.4643	2	0.0000	0.1235	0.9122	0	8	57
-0.4464	3	0.0000	0.1334	0.9202	0	8	58
-0.4286	4	0.0000	0.1436	0.9278	0	9	58
-0.4107	5	0.0000	0.1543	0.9350	0	10	59
-0.3929	6	0.0000	0.1653	0.9418	0	10	59
-0.3750	7	0.0000	0.1767	0.9483	0	11	60
-0.3571	8	0.0000	0.1885	0.9544	0	12	60
-0.3393	9	0.0000	0.2006	0.9601	0	13	60
-0.3214	10	0.0000	0.2132	0.9654	0	13	61
-0.3036	11	0.0000	0.2261	0.9703	0	14	61
-0.2857	12	0.0000	0.2394	0.9749	0	15	61
-0.2679	13	0.0000	0.2531	0.9791	0	16	62
-0.2500	14	0.0000	0.2671	0.9829	0	17	62
-0.2321	15	0.0000	0.2816	0.9863	0	18	62
-0.2143	16	0.0000	0.2964	0.9893	0	19	62
-0.1964	17	0.0000	0.3116	0.9920	0	20	62
-0.1786	18	0.0000	0.3272	0.9943	0	21	63
-0.1607	19	0.0000	0.3431	0.9962	0	22	63
-0.1429	20	0.0000	0.3595	0.9977	0	23	63
-0.1250	21	0.0000	0.3762	0.9988	0	24	63
-0.1071	22	0.0000	0.3933	0.9996	0	25	63
-0.0893	23	0.0000	0.4108	1.0000	0	26	63
-0.0714	24	0.0000	0.4286	0.0000	0	27	0
-0.0536	25	0.0000	0.4464	0.0000	0	28	0
-0.0357	26	0.0000	0.4643	0.0000	0	29	0
-0.0179	27	0.0000	0.4821	0.0000	0	30	0
-0.0000	28	0.0000	0.5000	0.0000	0	31	0
0.0179	29	0.0000	0.5179	0.0000	0	33	0
0.0357	30	0.0000	0.5357	0.0000	0	34	0
0.0536	31	0.0000	0.5536	0.0000	0	35	0
0.0714	32	0.0000	0.5714	0.0000	0	36	0
0.0893	33	0.0000	0.5892	0.0000	0	37	0
0.1071	34	0.0004	0.6067	0.0000	0	38	0
0.1250	35	0.0012	0.6238	0.0000	0	39	0
0.1429	36	0.0023	0.6405	0.0000	0	40	0
0.1607	37	0.0038	0.6569	0.0000	0	41	0
0.1786	38	0.0057	0.6728	0.0000	0	42	0
0.1964	39	0.0080	0.6884	0.0000	1	43	0
0.2143	40	0.0107	0.7036	0.0000	1	44	0
0.2321	41	0.0137	0.7184	0.0000	1	45	0
0.2500	42	0.0171	0.7329	0.0000	1	46	0
0.2679	43	0.0209	0.7469	0.0000	1	47	0
0.2857	44	0.0251	0.7606	0.0000	2	48	0
0.3036	45	0.0297	0.7739	0.0000	2	49	0
0.3214	46	0.0346	0.7868	0.0000	2	50	0
0.3393	47	0.0399	0.7994	0.0000	3	50	0
0.3571	48	0.0456	0.8115	0.0000	3	51	0
0.3750	49	0.0517	0.8233	0.0000	3	52	0
0.3929	50	0.0582	0.8347	0.0000	4	53	0
0.4107	51	0.0650	0.8457	0.0000	4	53	0
0.4286	52	0.0722	0.8564	0.0000	5	54	0
0.4464	53	0.0798	0.8666	0.0000	5	55	0
0.4643	54	0.0878	0.8765	0.0000	6	55	0
0.4821	55	0.0961	0.8860	0.0000	6	56	0
0.5000	56	0.1049	0.8951	0.0000	7	56	0

It is obvious that the area/intensity sampling look-up tables for lines and polygon edges in the first octant should be generated separately in increments of about 1° . These tables can also be used by the corresponded lines and edges in other octants with just a few minor modifications. For example, to anti-alias a line with a slope of 85° , the area/intensity sampling line look-up tables of 5° may be used but the horizontal distance instead of the vertical distance needs to be considered and the covered pixels are changed to be right, middle and left for each row rather than upper, middle and lower for each column. Slopes and distances of any real values can always be converted and rounded to integer values so that all lines and polygon edges can use suitable area/intensity sampling tables, get the area/intensity values precise enough for the covered pixels, and then be anti-aliased properly on the raster system.

Basically, the area/intensity sampling look-up tables can be used to anti-alias any kind of object image as long as there is no intersection of two or more objects. When that happens, the pixel area coverage and its intensity value at that intersection point have to be treated differently. The area/intensity sampling look-up tables not only allow the rapid removal of aliasing effects during object rendering but also allow the final images to be post-anti-aliased with just a little computation. The post-anti-aliasing technique is described in the next section.

(3) Display considerations. It is desirable that the intensity values of sampling look-up tables generated above have a linear relationship with the intensity values actually shown on the display in order to do anti-aliasing properly. However, some non-linear distortions often occur due to non-linearities of the display introduced by the device hardware, including the CRT phosphors and electronics, and the digital-to-analog converters. This effect can be almost as pronounced as the original aliasing. Previous methods of compensating these nonlinearities [Catm79] as well as Gamma correction discussed in Chapter 3 should be tailored to the chosen display device in order to remove nonlinearities.

Post-anti-aliasing

Based on varying period for different slope of each degree, i.e. 0° , 1° , 2° , ..., or 45° , as shown in Table 4-4, the area/intensity sampling look-up tables must be generated separately and in different sizes. Two examples have been shown in Table 4-5 and Table 4-6. The different sizes of area/intensity sampling look-up tables can really cause inconveniences in post-anti-aliasing. There is of course no reason to have tables for each degree of slope: one could just as easily use any other measure of the slope. What one needs is a measure that will yield the same periodicity for each of the slopes, so that for each slope the same number of post anti-aliasing entries are used. This will simplify the look-up tables into a single rectangular array. By dividing the first range of slopes, $[0,1]$, into 50 divisions instead of considering 45 integer degrees of slopes, we can get true periods for slopes of 0.02, 0.04, ..., and 1.00 as shown in Table 4-7 rather than just approximated-periodicity for slopes of integer degrees. And then by calculating the least common multiple of these true periods, we obtain the number 50, which is thus the fixed period for generating all area/intensity sampling look-up tables in the same size. Therefore, in this way, post-anti-aliasing can be done much easier.

From the division of the slope m , 50, the periodic length for the distance dx_y , 50, and the maximum number of covered pixels per column, 3, the dimension of all sampling look-up tables for post-anti-aliasing lines or polygon edges is $[51][51][3]$ and its type can be either real or integer. Storing precise real values for pixel area coverages is better whenever the background color of the final image must be considered and/or the number of anti-aliasing intensity levels is still flexible. On the other hand, since each integer value of intensity for each pixel needs only one byte of space, storing integer values for all of the covered pixels' intensities takes much less space and thus should be adopted if the background color is black, the number of anti-aliasing intensity levels is fixed, and speed of post-anti-aliasing as well as the storage space are concerns. These two types of tables

Table 4-7. Dividing the first range for slopes of lines and edges, [0,1], into 50 divisions.

Vector's Degree	Slope m => i		Multiple (Period)	Value (m*Period)	Closest Integer	Precision in Area	Precision in Intensity
0.000	0.000	0	1	0.0000	0	0.0000	0.0000
1.146	0.020	1	50	1.0000	1	0.0000	0.0000
2.291	0.040	2	25	1.0000	1	0.0000	0.0000
3.434	0.060	3	50	3.0000	3	0.0000	0.0000
4.574	0.080	4	25	2.0000	2	0.0000	0.0000
5.711	0.100	5	10	1.0000	1	0.0000	0.0000
6.843	0.120	6	25	3.0000	3	0.0000	0.0000
7.970	0.140	7	50	7.0000	7	0.0000	0.0000
9.090	0.160	8	25	4.0000	4	0.0000	0.0000
10.204	0.180	9	50	9.0000	9	0.0000	0.0000
11.310	0.200	10	5	1.0000	1	0.0000	0.0000
12.408	0.220	11	50	11.0000	11	0.0000	0.0000
13.496	0.240	12	25	6.0000	6	0.0000	0.0000
14.574	0.260	13	50	13.0000	13	0.0000	0.0000
15.642	0.280	14	25	7.0000	7	0.0000	0.0000
16.699	0.300	15	10	3.0000	3	0.0000	0.0000
17.745	0.320	16	25	8.0000	8	0.0000	0.0000
18.778	0.340	17	50	17.0000	17	0.0000	0.0000
19.799	0.360	18	25	9.0000	9	0.0000	0.0000
20.807	0.380	19	50	19.0000	19	0.0000	0.0000
21.802	0.400	20	5	2.0000	2	0.0000	0.0000
22.783	0.420	21	50	21.0000	21	0.0000	0.0000
23.750	0.440	22	25	11.0000	11	0.0000	0.0000
24.703	0.460	23	50	23.0000	23	0.0000	0.0000
25.641	0.480	24	25	12.0000	12	0.0000	0.0000
26.565	0.500	25	2	1.0000	1	0.0000	0.0000
27.475	0.520	26	25	13.0000	13	0.0000	0.0000
28.369	0.540	27	50	27.0000	27	0.0000	0.0001
29.249	0.560	28	25	14.0000	14	0.0000	0.0000
30.114	0.580	29	50	29.0000	29	0.0000	0.0000
30.964	0.600	30	5	3.0000	3	0.0000	0.0000
31.799	0.620	31	50	31.0000	31	0.0000	0.0000
32.620	0.640	32	25	16.0000	16	0.0000	0.0000
33.425	0.660	33	50	33.0000	33	0.0000	0.0000
34.216	0.680	34	25	17.0000	17	0.0000	0.0000
34.992	0.700	35	10	7.0000	7	0.0000	0.0000
35.754	0.720	36	25	18.0000	18	0.0000	0.0000
36.502	0.740	37	50	37.0000	37	0.0000	0.0000
37.235	0.760	38	25	19.0000	19	0.0000	0.0000
37.955	0.780	39	50	39.0000	39	0.0000	0.0000
38.660	0.800	40	5	4.0000	4	0.0000	0.0000
39.352	0.820	41	50	41.0000	41	0.0000	0.0000
40.031	0.840	42	25	21.0000	21	0.0000	0.0000
40.696	0.860	43	50	43.0000	43	0.0000	0.0000
41.348	0.880	44	25	22.0000	22	0.0000	0.0000
41.988	0.900	45	10	9.0000	9	0.0000	0.0000
42.614	0.920	46	25	23.0000	23	0.0000	0.0000
43.229	0.940	47	50	47.0000	47	0.0000	0.0000
43.831	0.960	48	25	24.0000	24	0.0000	0.0000
44.422	0.980	49	50	49.0000	49	0.0000	0.0000
45.000	1.000	50	1	1.0000	1	0.0000	0.0000

are so called area and intensity sampling look-up tables, respectively, and may both be generated and stored for use during post-anti-aliasing.

Table 4-8 shows the area/intensity sampling look-up tables for post-anti-aliasing a line with a slope of 0.080, i.e. 4.574° , to an approximation at 64 intensity levels. The slope m is rounded to a first integer index i to the look-up tables according to the formula $i = m * \text{divisions} + 0.5$, each distance dxy is also converted and rounded to a corresponding second integer index j according to the other formula $j = (dxy + 0.5) * \text{period} + 0.5$, where both of divisions and period are 50. The first index i here is 4 since $i = 0.080 * 50 + 0.5 \approx 4$. The close approximation of this line certainly includes the other line with a slope of 5° , i.e. 0.0875, because its corresponding integer index of slope, i , is 4 ($\approx 0.0875 * 50 + 0.5$) also. By comparing Table 4-8 with Table 4-5 at the same anti-aliasing intensity levels, 64, we find that the maximum difference between each of the covered pixel intensities of any distance dxy in Table 4-5 and of its corresponding integer index j in Table 4-8 is at most one intensity. Similarly, the same result can be obtained from comparison of Table 4-9, which is for anti-aliasing a polygon edge with a slope of 0.840, i.e. 40.031° , with Table 4-6, originally generated for anti-aliasing a polygon edge with a slope of 40° , i.e. 0.8391. A close examination shows that an accuracy of one pixel intensity level may be obtained with area/intensity sampling look-up tables having a dimension of [51][51][3] for both lines and polygon edges at 64 anti-aliasing intensity levels. For different intensity levels the size of tables may be changed. Also the area/intensity sampling look-up tables could be used to make anti-aliasing post-processors for providing fast and proper anti-aliasing features on raster display devices.

Finally we show some examples of the above techniques applied to post-anti-aliasing. Figure 4-18 is a post-anti-aliased image of a gridded surface in Figure 2-7, which shows aliased rendering of an object surface with partially ordered data. Figure 4-19 considers the background color of Figure 4-8 during anti-aliasing polygon edges in order to eliminate the problem of the appearance of incorrect color spots on the middle parts of edges of triangles.

Table 4-8. The area/intensity sampling look-up tables used to post-anti-alias a line with a slope of 0.080, i.e. 4.574° , at 64 intensity levels.

Slope m => i		Distance dxy => j		Pixel Area Coverage			64 Intensities		
				Upper	Middle	Lower	Up	Mid	Low
0.080	4	-0.5000	0	0.0000	0.5016	0.5016	0	32	32
0.080	4	-0.4800	1	0.0000	0.5216	0.4816	0	33	30
0.080	4	-0.4600	2	0.0000	0.5416	0.4616	0	34	29
0.080	4	-0.4400	3	0.0000	0.5616	0.4416	0	35	28
0.080	4	-0.4200	4	0.0000	0.5816	0.4216	0	37	27
0.080	4	-0.4000	5	0.0000	0.6016	0.4016	0	38	25
0.080	4	-0.3800	6	0.0000	0.6216	0.3816	0	39	24
0.080	4	-0.3600	7	0.0000	0.6416	0.3616	0	40	23
0.080	4	-0.3400	8	0.0000	0.6616	0.3416	0	42	22
0.080	4	-0.3200	9	0.0000	0.6816	0.3216	0	43	20
0.080	4	-0.3000	10	0.0000	0.7016	0.3016	0	44	19
0.080	4	-0.2800	11	0.0000	0.7216	0.2816	0	45	18
0.080	4	-0.2600	12	0.0000	0.7416	0.2616	0	47	16
0.080	4	-0.2400	13	0.0000	0.7616	0.2416	0	48	15
0.080	4	-0.2200	14	0.0000	0.7816	0.2216	0	49	14
0.080	4	-0.2000	15	0.0000	0.8016	0.2016	0	51	13
0.080	4	-0.1800	16	0.0000	0.8216	0.1816	0	52	11
0.080	4	-0.1600	17	0.0000	0.8416	0.1616	0	53	10
0.080	4	-0.1400	18	0.0000	0.8616	0.1416	0	54	9
0.080	4	-0.1200	19	0.0000	0.8816	0.1216	0	56	8
0.080	4	-0.1000	20	0.0000	0.9016	0.1016	0	57	6
0.080	4	-0.0800	21	0.0000	0.9216	0.0816	0	58	5
0.080	4	-0.0600	22	0.0000	0.9416	0.0616	0	59	4
0.080	4	-0.0400	23	0.0000	0.9616	0.0416	0	61	3
0.080	4	-0.0200	24	0.0029	0.9766	0.0237	0	62	1
0.080	4	0.0000	25	0.0108	0.9816	0.0108	1	62	1
0.080	4	0.0200	26	0.0237	0.9766	0.0029	1	62	0
0.080	4	0.0400	27	0.0416	0.9616	0.0000	3	61	0
0.080	4	0.0600	28	0.0616	0.9416	0.0000	4	59	0
0.080	4	0.0800	29	0.0816	0.9216	0.0000	5	58	0
0.080	4	0.1000	30	0.1016	0.9016	0.0000	6	57	0
0.080	4	0.1200	31	0.1216	0.8816	0.0000	8	56	0
0.080	4	0.1400	32	0.1416	0.8616	0.0000	9	54	0
0.080	4	0.1600	33	0.1616	0.8416	0.0000	10	53	0
0.080	4	0.1800	34	0.1816	0.8216	0.0000	11	52	0
0.080	4	0.2000	35	0.2016	0.8016	0.0000	13	51	0
0.080	4	0.2200	36	0.2216	0.7816	0.0000	14	49	0
0.080	4	0.2400	37	0.2416	0.7616	0.0000	15	48	0
0.080	4	0.2600	38	0.2616	0.7416	0.0000	16	47	0
0.080	4	0.2800	39	0.2816	0.7216	0.0000	18	45	0
0.080	4	0.3000	40	0.3016	0.7016	0.0000	19	44	0
0.080	4	0.3200	41	0.3216	0.6816	0.0000	20	43	0
0.080	4	0.3400	42	0.3416	0.6616	0.0000	22	42	0
0.080	4	0.3600	43	0.3616	0.6416	0.0000	23	40	0
0.080	4	0.3800	44	0.3816	0.6216	0.0000	24	39	0
0.080	4	0.4000	45	0.4016	0.6016	0.0000	25	38	0
0.080	4	0.4200	46	0.4216	0.5816	0.0000	27	37	0
0.080	4	0.4400	47	0.4416	0.5616	0.0000	28	35	0
0.080	4	0.4600	48	0.4616	0.5416	0.0000	29	34	0
0.080	4	0.4800	49	0.4816	0.5216	0.0000	30	33	0
0.080	4	0.5000	50	0.5016	0.5016	0.0000	32	32	0

Table 4-9. The area/intensity sampling look-up tables for post-anti-aliasing a polygon edge with flag 1 and a slope of 0.840, i.e. 40.031° , at 64 intensity levels.

Slope m => i		Distance dxy => j		Pixel Area Coverage			64 Intensities		
				Upper	Middle	Lower	Up	Mid	Low
0.840	42	-0.500	0	0.0000	0.1050	0.8950	0	7	56
0.840	42	-0.480	1	0.0000	0.1152	0.9048	0	7	57
0.840	42	-0.460	2	0.0000	0.1260	0.9140	0	8	58
0.840	42	-0.440	3	0.0000	0.1371	0.9229	0	9	58
0.840	42	-0.420	4	0.0000	0.1488	0.9312	0	9	59
0.840	42	-0.400	5	0.0000	0.1610	0.9390	0	10	59
0.840	42	-0.380	6	0.0000	0.1736	0.9464	0	11	60
0.840	42	-0.360	7	0.0000	0.1867	0.9533	0	12	60
0.840	42	-0.340	8	0.0000	0.2002	0.9598	0	13	60
0.840	42	-0.320	9	0.0000	0.2143	0.9657	0	14	61
0.840	42	-0.300	10	0.0000	0.2288	0.9712	0	14	61
0.840	42	-0.280	11	0.0000	0.2438	0.9762	0	15	61
0.840	42	-0.260	12	0.0000	0.2593	0.9807	0	16	62
0.840	42	-0.240	13	0.0000	0.2752	0.9848	0	17	62
0.840	42	-0.220	14	0.0000	0.2917	0.9883	0	18	62
0.840	42	-0.200	15	0.0000	0.3086	0.9914	0	19	62
0.840	42	-0.180	16	0.0000	0.3260	0.9940	0	21	63
0.840	42	-0.160	17	0.0000	0.3438	0.9962	0	22	63
0.840	42	-0.140	18	0.0000	0.3621	0.9979	0	23	63
0.840	42	-0.120	19	0.0000	0.3810	0.9990	0	24	63
0.840	42	-0.100	20	0.0000	0.4002	0.9998	0	25	63
0.840	42	-0.080	21	0.0000	0.4200	1.0000	0	26	63
0.840	42	-0.060	22	0.0000	0.4400	0.0000	0	28	0
0.840	42	-0.040	23	0.0000	0.4600	0.0000	0	29	0
0.840	42	-0.020	24	0.0000	0.4800	0.0000	0	30	0
0.840	42	0.000	25	0.0000	0.5000	0.0000	0	32	0
0.840	42	0.020	26	0.0000	0.5200	0.0000	0	33	0
0.840	42	0.040	27	0.0000	0.5400	0.0000	0	34	0
0.840	42	0.060	28	0.0000	0.5600	0.0000	0	35	0
0.840	42	0.080	29	0.0000	0.5800	0.0000	0	37	0
0.840	42	0.100	30	0.0002	0.5998	0.0000	0	38	0
0.840	42	0.120	31	0.0010	0.6190	0.0000	0	39	0
0.840	42	0.140	32	0.0021	0.6379	0.0000	0	40	0
0.840	42	0.160	33	0.0038	0.6562	0.0000	0	41	0
0.840	42	0.180	34	0.0060	0.6740	0.0000	0	42	0
0.840	42	0.200	35	0.0086	0.6914	0.0000	1	44	0
0.840	42	0.220	36	0.0117	0.7083	0.0000	1	45	0
0.840	42	0.240	37	0.0152	0.7248	0.0000	1	46	0
0.840	42	0.260	38	0.0193	0.7407	0.0000	1	47	0
0.840	42	0.280	39	0.0238	0.7562	0.0000	2	48	0
0.840	42	0.300	40	0.0288	0.7712	0.0000	2	49	0
0.840	42	0.320	41	0.0343	0.7857	0.0000	2	49	0
0.840	42	0.340	42	0.0402	0.7998	0.0000	3	50	0
0.840	42	0.360	43	0.0467	0.8133	0.0000	3	51	0
0.840	42	0.380	44	0.0536	0.8264	0.0000	3	52	0
0.840	42	0.400	45	0.0610	0.8390	0.0000	4	53	0
0.840	42	0.420	46	0.0688	0.8512	0.0000	4	54	0
0.840	42	0.440	47	0.0771	0.8629	0.0000	5	54	0
0.840	42	0.460	48	0.0860	0.8740	0.0000	5	55	0
0.840	42	0.480	49	0.0952	0.8848	0.0000	6	56	0
0.840	42	0.500	50	0.1050	0.8950	0.0000	7	56	0

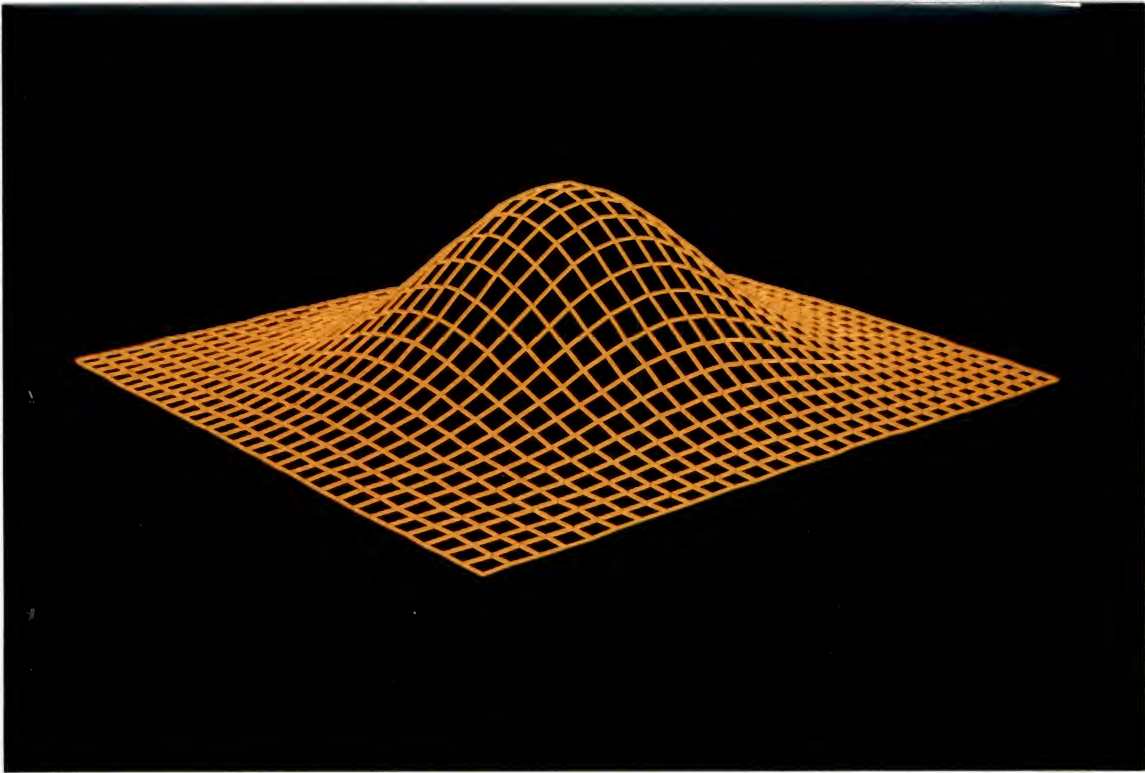


Figure 4-18. A post-anti-aliased image of Figure 2-7.

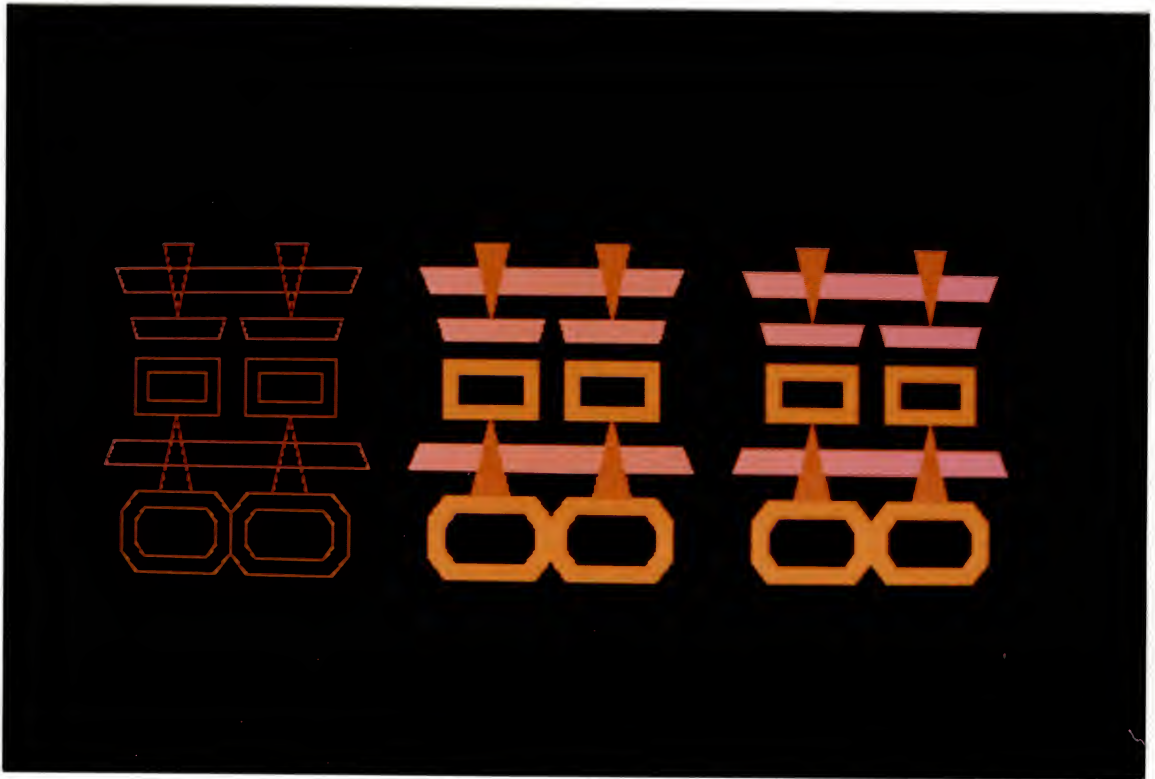


Figure 4-19. Figure 4-8 with correct background color calculations.

Figure 4-20 shows an aliased molecular model of the gelation point presented by Staudhammer et al. [Stau87, Stau88], which is post-anti-aliased in Figure 4-21. Figure 4-22 shows the same gelation molecular model but with a blue background instead of black, a different color look-up table, and also a different viewing angle.

The computation costs of the post-anti-aliasing using the Macintosh II in the various pictures shown above are:

Pictures	# of Columns/Rows	# of Pixels	Execution Time (seconds)
Figure 4-18	13967	34007	0.92
Figure 4-19 (Right)	2364	2504	0.13
Figure 4-21	8544	11748	0.45
Figure 4-22	8544	11748	0.58

where # stands for "number", the # of columns/rows and the # of pixels indicate the space that the picture silhouette takes on a raster display. Here, we count the time of calculating the slope m for each line (or edge) and the distance dxy for each column (or row), of converting them to the first and the second indices, i and j , of accessing the area (or intensity) values in the area (or intensity) sampling look-up tables and of computing the suitable intensity values of color components (red, green, blue) for each anti-aliased pixel and its position on a raster device with correct background color calculations. Sutherland mentioned in an interview [Fren89] that most of what is in the computer is wires and very small fraction of the computer is switches, so an adequate theory to deal with computing has to involve the communication of numbers from one place to another. In fact, communication is expensive and arithmetic is relatively cheap. The computation costs counted above included both of them and these costs could further be minimized by reducing the amount of communication of variables in the programs.

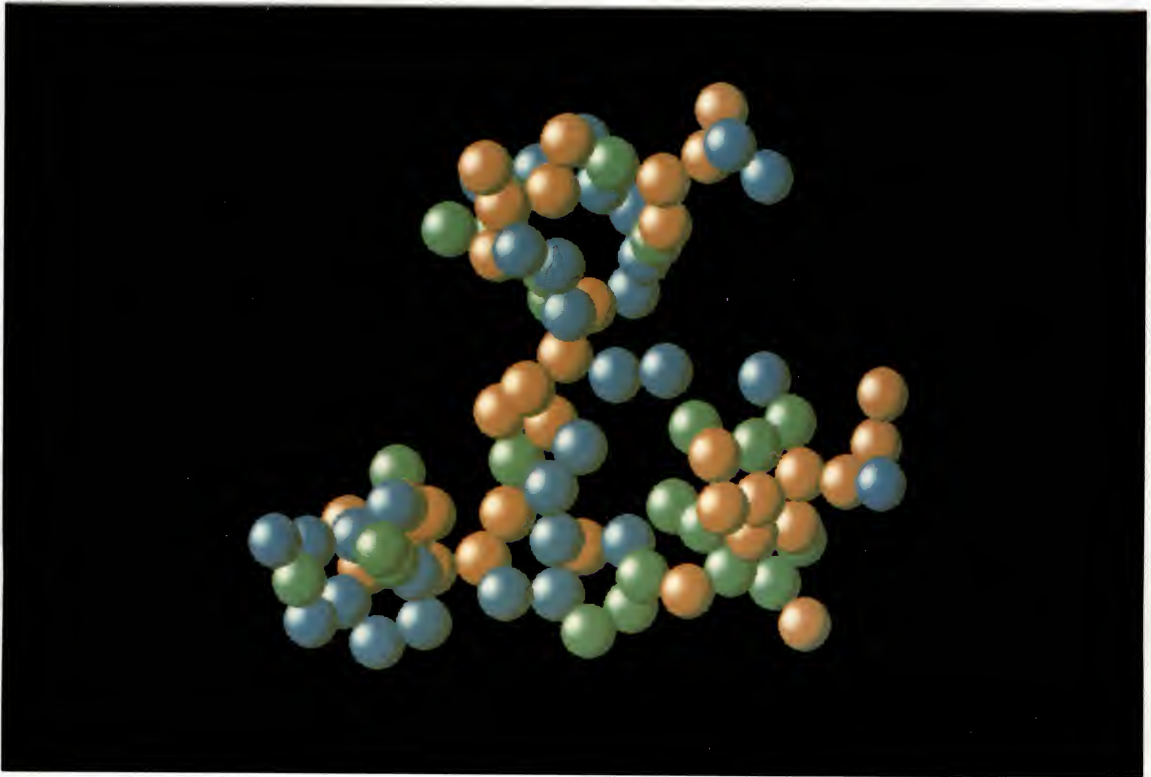


Figure 4-20. A molecular model of the gelation point; note aliasing effects.

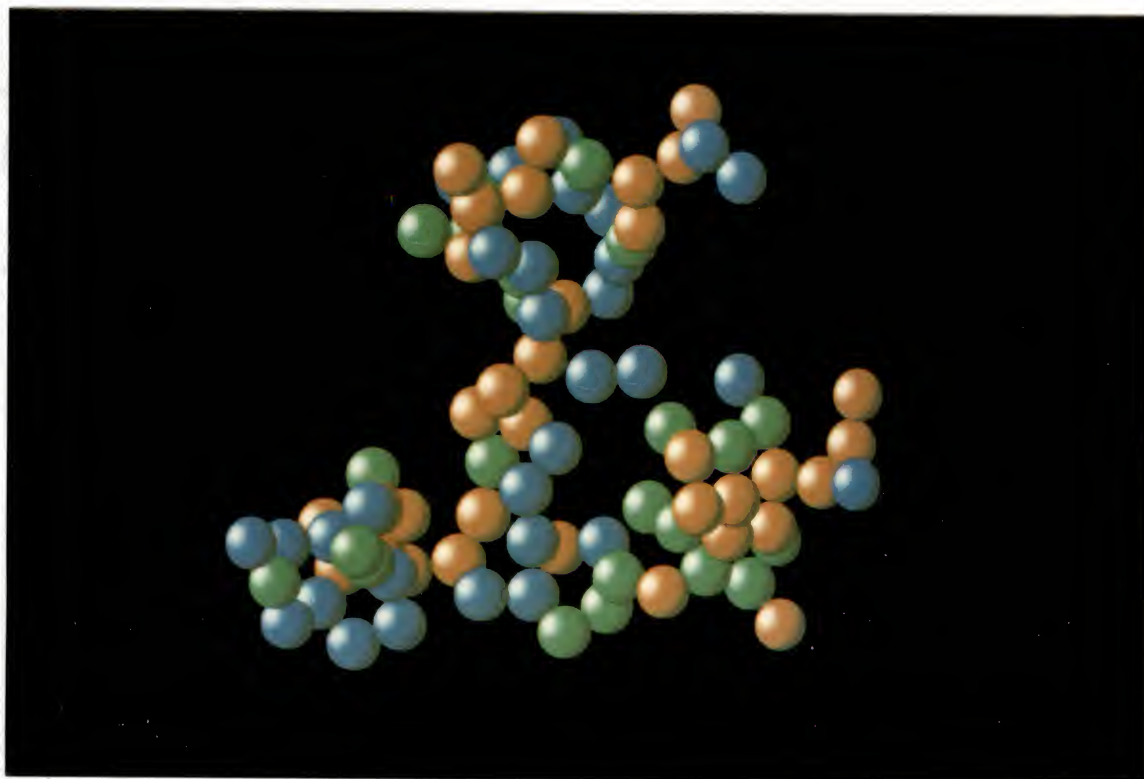


Figure 4-21. A post-anti-aliased molecular model of the gelation point.

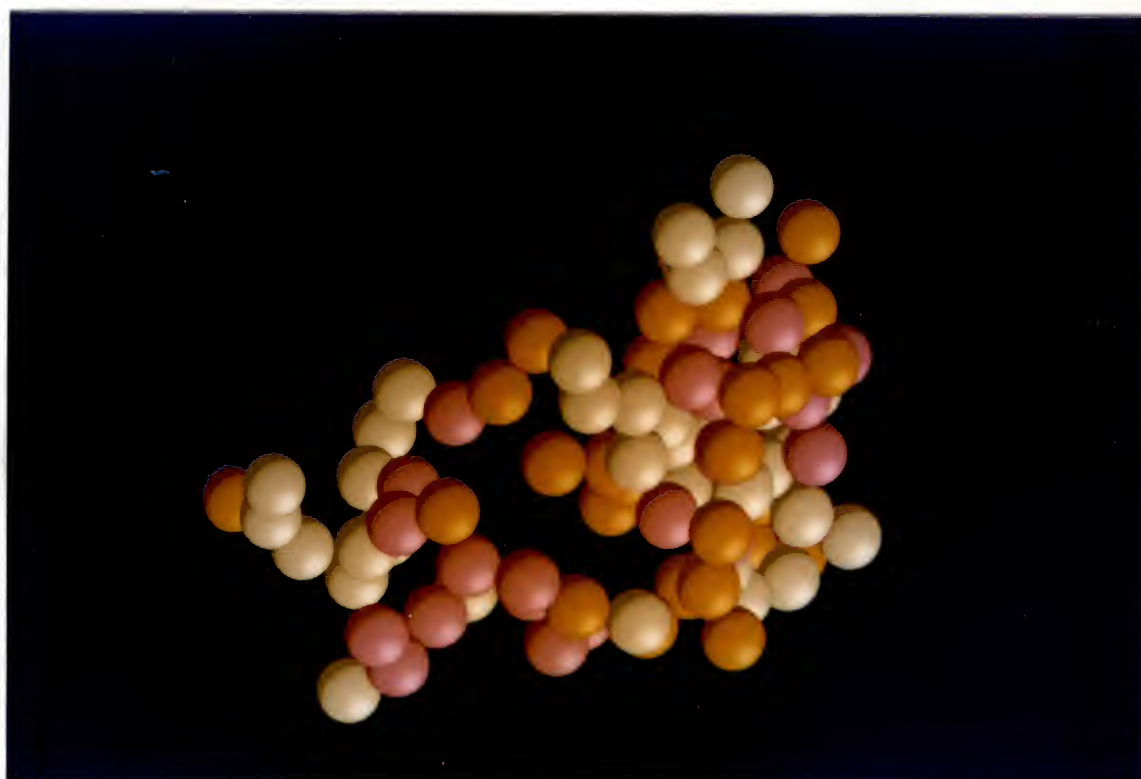


Figure 4-22. Another post-anti-aliased molecular model of the gelation point with a blue background, a different color look-up table, and a different viewing angle.

CHAPTER 5 EXTENSIONS OF THE ALGORITHMS

The algorithms of visibility determination, shading and anti-aliasing that were developed and described in the previous chapters may be extended for image generation of solid objects defined by partially ordered surface data with hidden-line/surface elimination and anti-aliasing/post-anti-aliasing. The images may be drawn with anti-aliased lines or smoothed shaded surfaces; these two kinds of algorithms are discussed in this chapter.

A Hidden-line Algorithm with Anti-aliasing/Post-anti-aliasing

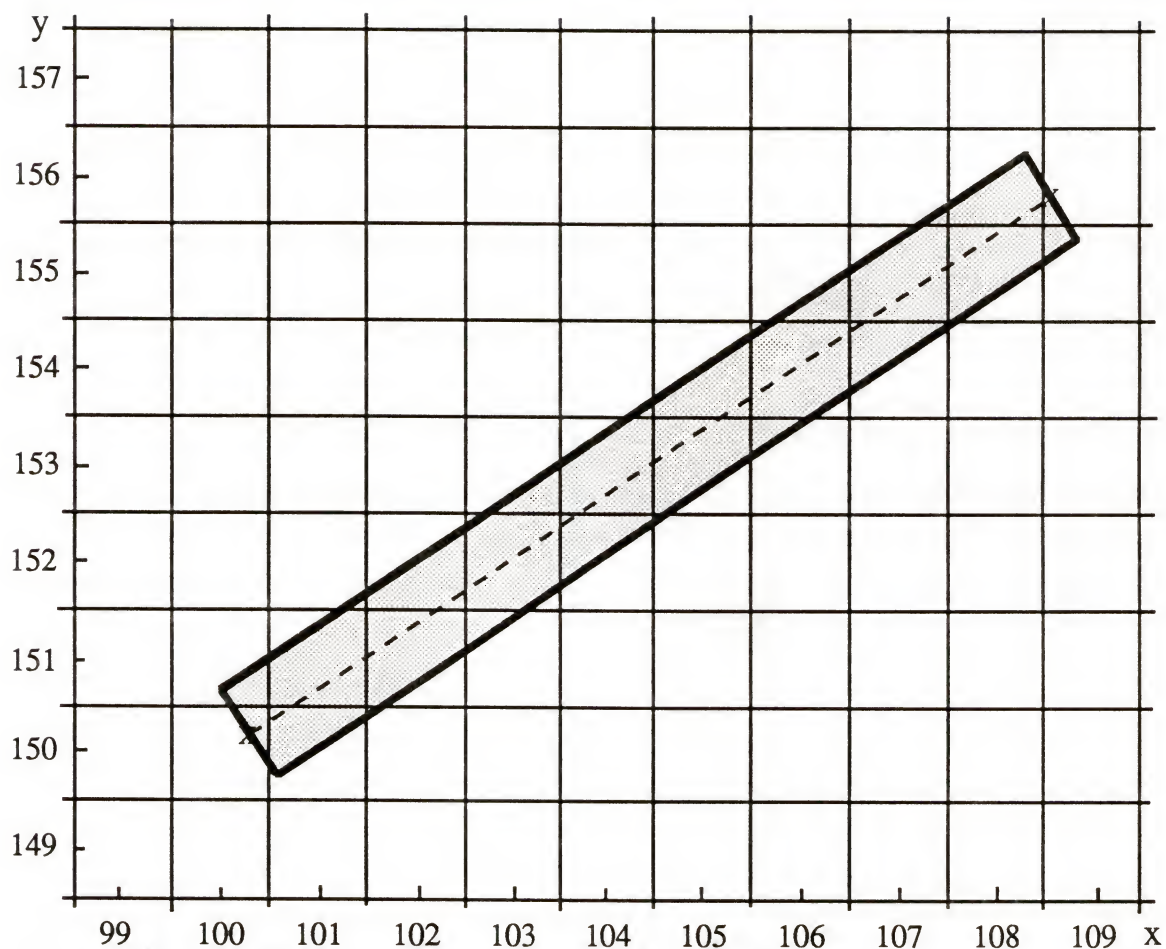
The Floating Perimeter Algorithm produces all visible line segments for the image generation of algebraic surfaces. These visible line segments can be displayed as constant-intensity smooth lines by directly using the anti-aliasing line algorithm DISLOP_L or they can be approximately smoothed by post-anti-aliasing with the closest area/intensity sampling look-up table. An image data file can be created for storing two pairs of end points in floating point numbers for each of all visible line segments of a surface image. The format of the file is very simple as shown in the following:

```
x1s  y1s   x1e  y1e
x2s  y2s   x2e  y2e
.....
xns  yns   xne  yne
```

where n is the total number of visible line segments for the surface image, s indicates the start point of a line segment, and e indicates its end point. To use the algorithm DISLOP_L for anti-aliasing each line segment, we simply pass both end points to it. The algorithm will produce each pixel coverage in all columns/rows traversed by the line segment along with

the pixel's X & Y-screen coordinates. For example, suppose the 1st line segment has end points: $x_{1s} = 100.3$, $y_{1s} = 150.2$, $x_{1e} = 108.6$, and $y_{1e} = 155.8$. We obtain an anti-aliased line segment with covered areas of all passed pixels as shown in Figure 5-1. Now, one thing we left so far is the intensity of the line segment, which is related to the intensity of the surface image. Assume the object surface has uniform intensity with the maximum level 255 throughout the entire surface, then all its visible line segments should have intensity 255 also. Taking the intensity into account, the algorithm DISLOP_L can directly produce integer intensity values for all pixels covered by each line segment instead of the original results, the covered areas with floating point precision. Thus this needs much less computation than modulating the light intensity with the area coverages for displaying the entire surface image with visible smoothed lines.

Another approach for obtaining an anti-aliased visible-line surface image is to use the area/intensity sampling look-up table, in which values have been generated by the algorithm DISLOP_L and stored for recall. As mentioned in the section of post-anti-aliasing in the last chapter, we adopt 50 divisions for the slope m in $[0, 1]$, 50 columns for the periodic length of the distance dxy , and 3 pixels for the maximum number of pixels covered by a line segment per column, so that the area/intensity sampling look-up table has a dimension of $[51][51][3]$ for all visible line segments. Suppose the background color of the final image is black and the chosen intensity level is 64, the intensity sampling look-up table would contain integer intensity values in the range of $[0, 63]$ for all elements. Part of this table was shown in Figure 4-8 for post-anti-aliasing a line with a slope $m = 0.080$ or its close approximation. As for the above example, the line segment has a slope $m \approx 0.6747$ (see Figure 5-2), which should be converted and rounded to an integer index i , 34, therefore the part of the area/intensity sampling look-up table suitable for the middle section of this line segment would be that shown in Table 5-1. Note that the examples in this work use an intensity accuracy of 1:64. Although all table entries are printed in floating point precision, only a 2% accuracy is used in the final values. Since the initial vertical distance



x	y	Area	x	y	Area
100	151	0.128100	105	154	0.474247
100	150	0.120004	105	153	0.728506
			105	152	0.003572
101	152	0.009429	106	155	0.175252
101	151	0.766019	106	154	0.915607
101	150	0.424035	106	153	0.115466
			107	156	0.019209
102	152	0.450147	107	155	0.804438
102	151	0.749695	107	154	0.382678
102	150	0.006482			
103	153	0.158312	108	156	0.463541
103	152	0.918017	108	155	0.706456
103	151	0.129996	108	154	0.001523
			109	156	0.067772
104	154	0.013888	109	155	0.087664
104	153	0.785659			
104	152	0.406778			

Figure 5-1. Pixel area coverage for the 1-pixel wide anti-aliased line segment with two ends (100.3,150.2) and (108.6,155.8) using the algorithm DISLOP_L.

$$m = \frac{(y_{1e} - y_{1s})}{(x_{1e} - x_{1s})} = \frac{(155.8 - 150.2)}{(108.6 - 100.3)} = 0.6747$$

$$i = m * \text{divisions} + 0.5 = 0.6747 * 50 + 0.5 = 34.235 \approx 34$$

$$x_0 = 100$$

$$y_0 = 150.2 - (100.3 - 100) * m = 149.9976$$

From Figure 4-1:

$$dxy_0 = y_0 - IY_0 = 149.9976 - 150 = -0.0024$$

Therefore,

$$j_0 = (dxy_0 + 0.5) * \text{period} + 0.5 = (-0.0024 + 0.5) * 50 + 0.5 = 25.38 \approx 25$$

x	k	dxy _k	j _k	j _k '
	(Column #)	(= dxy _{k-1} + m (-1 or not))	(= (dxy _k +0.5)*period+0.5)	(= (j _{k-1} ' + i) % 50)
100	0	-0.0024	25	25
101	1	-0.3277 (= -0.0024+0.6747-1)	9 (= (-0.3277+0.5)*50+0.5)	9 (= (25+34)%50)
102	2	0.3470 (= -0.3277+0.6747)	42 (= (0.3470+0.5)*50+0.5)	43 (= (9+34)%50)
103	3	0.0217 (= 0.3470+0.6747-1)	26 (= (0.0217+0.5)*50+0.5)	27 (= (43+34)%50)
104	4	-0.3036 (= 0.0217+0.674 -1)	10 (= (-0.3036+0.5)*50+0.5)	11 (= (27+34)%50)
105	5	0.3711 (= -0.3036+0.6747)	44 (= (0.3711+0.5)*50+0.5)	45 (= (11+34)%50)
106	6	0.0458 (= 0.3711+0.6747-1)	27 (= (0.0458+0.5)*50+0.5)	29 (= (45+34)%50)
107	7	-0.2795 (= 0.0458+0.6747-1)	11 (= (-0.2795+0.5)*50+0.5)	13 (= (29+34)%50)
108	8	0.3952 (= -0.2795+0.6747)	45 (= (0.3952+0.5)*50+0.5)	47 (= (13+34)%50)

Figure 5-2. Calculation of indices to the area/intensity sampling look-up tables for the line segment in Figure 5-1. The approximate calculation of the second index j_k' is also shown.

Table 5-1. The area/intensity sampling look-up tables used for post-anti-aliasing a line with a slope of 0.680, i.e. 34.216° at 64 intensity levels.

Slope m => i		Distance dxy => j		Pixel Area Coverage			64 Intensities		
				Upper	Middle	Lower	Up	Mid	Low
0.680	34	-0.500	0	0.0000	0.6046	0.6046	0	38	38
0.680	34	-0.480	1	0.0000	0.6246	0.5846	0	39	37
0.680	34	-0.460	2	0.0000	0.6446	0.5646	0	41	36
0.680	34	-0.440	3	0.0000	0.6646	0.5446	0	42	34
0.680	34	-0.420	4	0.0004	0.6842	0.5246	0	43	33
0.680	34	-0.400	5	0.0015	0.7032	0.5046	0	44	32
0.680	34	-0.380	6	0.0031	0.7216	0.4846	0	45	31
0.680	34	-0.360	7	0.0053	0.7394	0.4646	0	47	29
0.680	34	-0.340	8	0.0081	0.7566	0.4446	1	48	28
0.680	34	-0.320	9	0.0114	0.7732	0.4246	1	49	27
0.680	34	-0.300	10	0.0154	0.7893	0.4046	1	50	25
0.680	34	-0.280	11	0.0199	0.8047	0.3846	1	51	24
0.680	34	-0.260	12	0.0251	0.8196	0.3646	2	52	23
0.680	34	-0.240	13	0.0308	0.8339	0.3446	2	53	22
0.680	34	-0.220	14	0.0371	0.8474	0.3248	2	53	20
0.680	34	-0.200	15	0.0440	0.8597	0.3056	3	54	19
0.680	34	-0.180	16	0.0515	0.8709	0.2869	3	55	18
0.680	34	-0.160	17	0.0596	0.8809	0.2688	4	55	17
0.680	34	-0.140	18	0.0682	0.8897	0.2513	4	56	16
0.680	34	-0.120	19	0.0775	0.8974	0.2344	5	57	15
0.680	34	-0.100	20	0.0873	0.9038	0.2181	6	57	14
0.680	34	-0.080	21	0.0978	0.9091	0.2024	6	57	13
0.680	34	-0.060	22	0.1088	0.9132	0.1873	7	58	12
0.680	34	-0.040	23	0.1204	0.9162	0.1727	8	58	11
0.680	34	-0.020	24	0.1326	0.9180	0.1587	8	58	10
0.680	34	0.000	25	0.1454	0.9185	0.1454	9	58	9
0.680	34	0.020	26	0.1587	0.9180	0.1326	10	58	8
0.680	34	0.040	27	0.1727	0.9162	0.1204	11	58	8
0.680	34	0.060	28	0.1873	0.9132	0.1088	12	58	7
0.680	34	0.080	29	0.2024	0.9091	0.0978	13	57	6
0.680	34	0.100	30	0.2181	0.9038	0.0873	14	57	6
0.680	34	0.120	31	0.2344	0.8974	0.0775	15	57	5
0.680	34	0.140	32	0.2513	0.8897	0.0682	16	56	4
0.680	34	0.160	33	0.2688	0.8809	0.0596	17	55	4
0.680	34	0.180	34	0.2869	0.8709	0.0515	18	55	3
0.680	34	0.200	35	0.3056	0.8597	0.0440	19	54	3
0.680	34	0.220	36	0.3248	0.8474	0.0371	20	53	2
0.680	34	0.240	37	0.3446	0.8339	0.0308	22	53	2
0.680	34	0.260	38	0.3646	0.8196	0.0251	23	52	2
0.680	34	0.280	39	0.3846	0.8047	0.0199	24	51	1
0.680	34	0.300	40	0.4046	0.7893	0.0154	25	50	1
0.680	34	0.320	41	0.4246	0.7732	0.0114	27	49	1
0.680	34	0.340	42	0.4446	0.7566	0.0081	28	48	1
0.680	34	0.360	43	0.4646	0.7394	0.0053	29	47	0
0.680	34	0.380	44	0.4846	0.7216	0.0031	31	45	0
0.680	34	0.400	45	0.5046	0.7032	0.0015	32	44	0
0.680	34	0.420	46	0.5246	0.6842	0.0004	33	43	0
0.680	34	0.440	47	0.5446	0.6646	0.0000	34	42	0
0.680	34	0.460	48	0.5646	0.6446	0.0000	36	41	0
0.680	34	0.480	49	0.5846	0.6246	0.0000	37	39	0
0.680	34	0.500	50	0.6046	0.6046	0.0000	38	38	0

dxy_0 is -0.0024, all the other distances can be obtained by simply adding the slope m to their corresponding previous distances, i.e. $dxy_k = dxy_{k-1} + m$, where k is the column/row number of pixels passed by the line segment. However, the value of dxy_k must be subtracted from 1 whenever it is greater than its upper limit 0.5. Then, using the formula $(dxy_k + 0.5) * \text{period} + 0.5 = j_k$, each distance dxy_k can be converted and rounded to a second integer index j_k to the area/intensity sampling look-up table in Table 5-1. Another simpler calculation of the second index is that $j_k' = j_{k-1}' + i$ and then $j_k' = j_k' \% 50$, i.e. the value of j_k' is always the remainder of itself being divided by 50. These two sets of indices j_k and j_{k-1}' are calculated in Figure 5-2, which are then used to refer to the covered pixels' areas or the intensity values in the area/intensity sampling look-up table as shown in Table 5-2 for each of all columns passed by the middle section of the line segment. By comparing each of the covered pixels' areas in the column #4 and #4 in Table 5-2 (which, in this case, refer to two different second indices $j_4 (=10)$ and $j_4' (=11)$, respectively) with each of their corresponding covered pixels' areas in the column $x=104$ (#4) in Figure 5-1, we note that the first set of the second indices j_k is more accurate for table look-up.

What are the exact differences between anti-aliasing and post-anti-aliasing? Perhaps, they can be described from two aspects: computation complexity and image quality. Using the anti-aliasing algorithm DISLOP_L to directly calculate intensity values of pixels passed needs more computation time than just calculating integer indices for looking into the area/intensity sampling tables. As for image quality, it really depends on how smooth the image needs to be and how the chosen raster display behaves. For the same object surface displayed in the top and the bottom of Figure 5-3 using two different techniques, can we detect any difference between them? By comparing the whole smoothness of these two gridded surface images and the local smoothness of each pair of corresponding line segment, we see that only a very little difference exists. Therefore, utilizing the area/intensity sampling look-up table for post-anti-aliasing can greatly simplify the computations and still produce a relatively high quality image. Other examples of post-anti-

Table 5-2. The two sets of second indices j_k and j_{k-1} used to look into the area and intensity sampling tables.

Slope m=>i		Distance dxy =>j		Column #		Pixel Area Coverage			64 Intensities		
						Upper	Middle	Lower	Up	Mid	Low
0.680	34	-0.500	0			0.0000	0.6046	0.6046	0	38	38
0.680	34	-0.480	1			0.0000	0.6246	0.5846	0	39	37
0.680	34	-0.460	2			0.0000	0.6446	0.5646	0	41	36
0.680	34	-0.440	3			0.0000	0.6646	0.5446	0	42	34
0.680	34	-0.420	4			0.0004	0.6842	0.5246	0	43	33
0.680	34	-0.400	5			0.0015	0.7032	0.5046	0	44	32
0.680	34	-0.380	6			0.0031	0.7216	0.4846	0	45	31
0.680	34	-0.360	7			0.0053	0.7394	0.4646	0	47	29
0.680	34	-0.340	8			0.0081	0.7566	0.4446	1	48	28
0.680	34	-0.320	9	1	1	0.0114	0.7732	0.4246	1	49	27
0.680	34	-0.300	10	4		0.0154	0.7893	0.4046	1	50	25
0.680	34	-0.280	11	7		0.0199	0.8047	0.3846	1	51	24
0.680	34	-0.260	12			0.0251	0.8196	0.3646	2	52	23
0.680	34	-0.240	13			0.0308	0.8339	0.3446	2	53	22
0.680	34	-0.220	14			0.0371	0.8474	0.3248	2	53	20
0.680	34	-0.200	15			0.0440	0.8597	0.3056	3	54	19
0.680	34	-0.180	16			0.0515	0.8709	0.2869	3	55	18
0.680	34	-0.160	17			0.0596	0.8809	0.2688	4	55	17
0.680	34	-0.140	18			0.0682	0.8897	0.2513	4	56	16
0.680	34	-0.120	19			0.0775	0.8974	0.2344	5	57	15
0.680	34	-0.100	20			0.0873	0.9038	0.2181	6	57	14
0.680	34	-0.080	21			0.0978	0.9091	0.2024	6	57	13
0.680	34	-0.060	22			0.1088	0.9132	0.1873	7	58	12
0.680	34	-0.040	23			0.1204	0.9162	0.1727	8	58	11
0.680	34	-0.020	24			0.1326	0.9180	0.1587	8	58	10
0.680	34	0.000	25			0.1454	0.9185	0.1454	9	58	9
0.680	34	0.020	26	3		0.1587	0.9180	0.1326	10	58	8
0.680	34	0.040	27	6	3	0.1727	0.9162	0.1204	11	58	8
0.680	34	0.060	28			0.1873	0.9132	0.1088	12	58	7
0.680	34	0.080	29		6	0.2024	0.9091	0.0978	13	57	6
0.680	34	0.100	30			0.2181	0.9038	0.0873	14	57	6
0.680	34	0.120	31			0.2344	0.8974	0.0775	15	57	5
0.680	34	0.140	32			0.2513	0.8897	0.0682	16	56	4
0.680	34	0.160	33			0.2688	0.8809	0.0596	17	55	4
0.680	34	0.180	34			0.2869	0.8709	0.0515	18	55	3
0.680	34	0.200	35			0.3056	0.8597	0.0440	19	54	3
0.680	34	0.220	36			0.3248	0.8474	0.0371	20	53	2
0.680	34	0.240	37			0.3446	0.8339	0.0308	22	53	2
0.680	34	0.260	38			0.3646	0.8196	0.0251	23	52	2
0.680	34	0.280	39			0.3846	0.8047	0.0199	24	51	1
0.680	34	0.300	40			0.4046	0.7893	0.0154	25	50	1
0.680	34	0.320	41			0.4246	0.7732	0.0114	27	49	1
0.680	34	0.340	42	2		0.4446	0.7566	0.0081	28	48	1
0.680	34	0.360	43		2	0.4646	0.7394	0.0053	29	47	0
0.680	34	0.380	44	5		0.4846	0.7216	0.0031	31	45	0
0.680	34	0.400	45	8	5	0.5046	0.7032	0.0015	32	44	0
0.680	34	0.420	46			0.5246	0.6842	0.0004	33	43	0
0.680	34	0.440	47		8	0.5446	0.6646	0.0000	34	42	0
0.680	34	0.460	48			0.5646	0.6446	0.0000	36	41	0
0.680	34	0.480	49			0.5846	0.6246	0.0000	37	39	0
0.680	34	0.500	50			0.6046	0.6046	0.0000	38	38	0

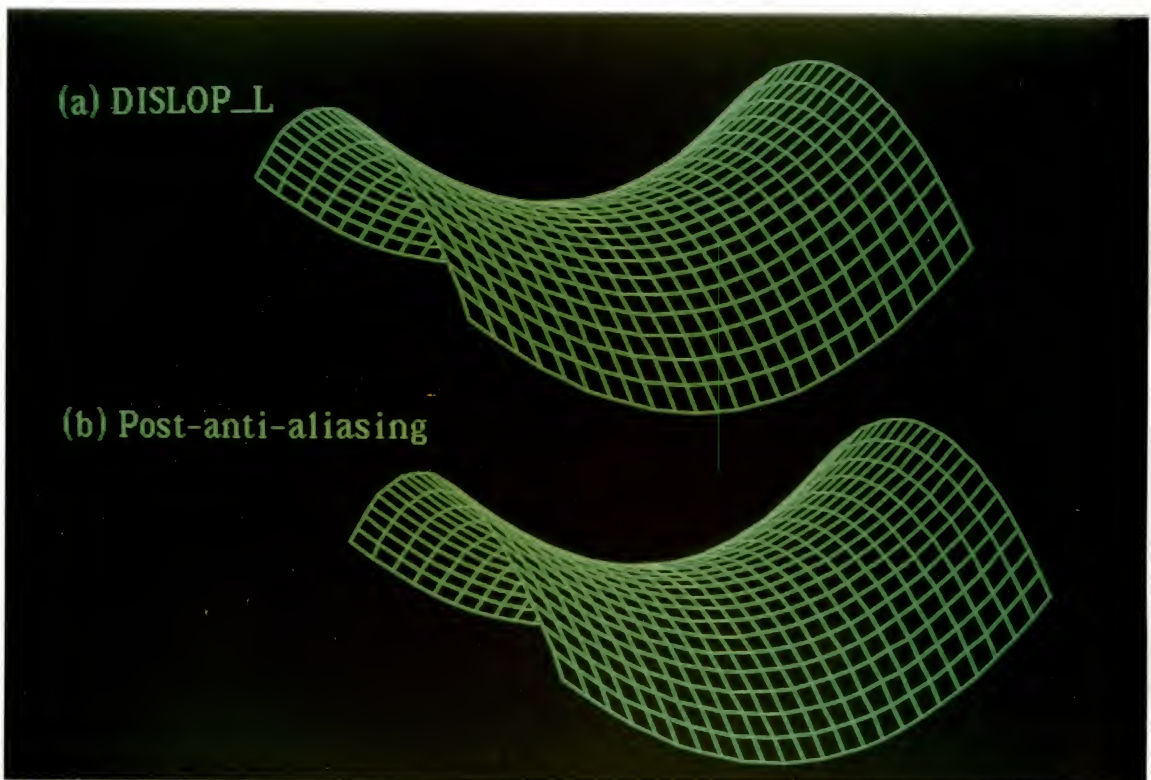


Figure 5-3. Smoothed gridded surface images of a polynomial function using (a) the anti-aliasing algorithm DISLOP_L and (b) the intensity sampling look-up table for post-anti-aliasing.
Note: the same color was used for both images in an effort to minimize the effects of color-gun differences.

aliased gridded surface images by means of the area/intensity sampling look-up table along with the consideration of non-black background are shown in Figure 5-4, which is a surface of a Bessel function of the first kind, and in Figure 5-5, a sine wave function surface.

Smoothed Visible Surfaces

The surface images in Figure 3-4 and 3-5 show that their shading and high-lighting are very smooth, but their silhouettes are noticeably jagged and should be anti-aliased to improve the quality of the image. Since improving the rendering of the silhouette edges of these images is similar to smoothing line segments, there are also two techniques which can be applied. One is to use the anti-aliasing edge algorithm DISLOP_E, the other is to post-anti-alias edges with area/intensity sampling look-up tables. The differences between these two approaches are also similar to those for anti-aliasing lines. Therefore, the latter approach will have less computation complexity.

The area/intensity sampling look-up tables for edges need to be generated for two kinds of flag assignments, i.e. edges with flag 0 and flag 1 (see Figure 4-7). The area/intensity values in the tables are actually calculated using the anti-aliasing edge algorithm DISLOP_E and then stored for later use. For anti-aliasing a common edge, as shown in Figure 5-6, with two ends (100.3,150.2) and (108.6,155.8) on the bottom of a polygon (edge flag-0) and on the top of the other (edge flag-1) at 64 intensity levels, the parts of these two area/intensity sampling look-up tables for this edge are completely different and are shown in Table 5-3 and 5-4, respectively. Therefore, instead of just one area/intensity sampling look-up table for smoothing line segments described in the previous section, two area/intensity sampling look-up tables are required for post-anti-aliasing polygon edges or silhouette edges. However, the methods of calculating the first and second indices to the look-up tables for edges are just the same as those for lines.

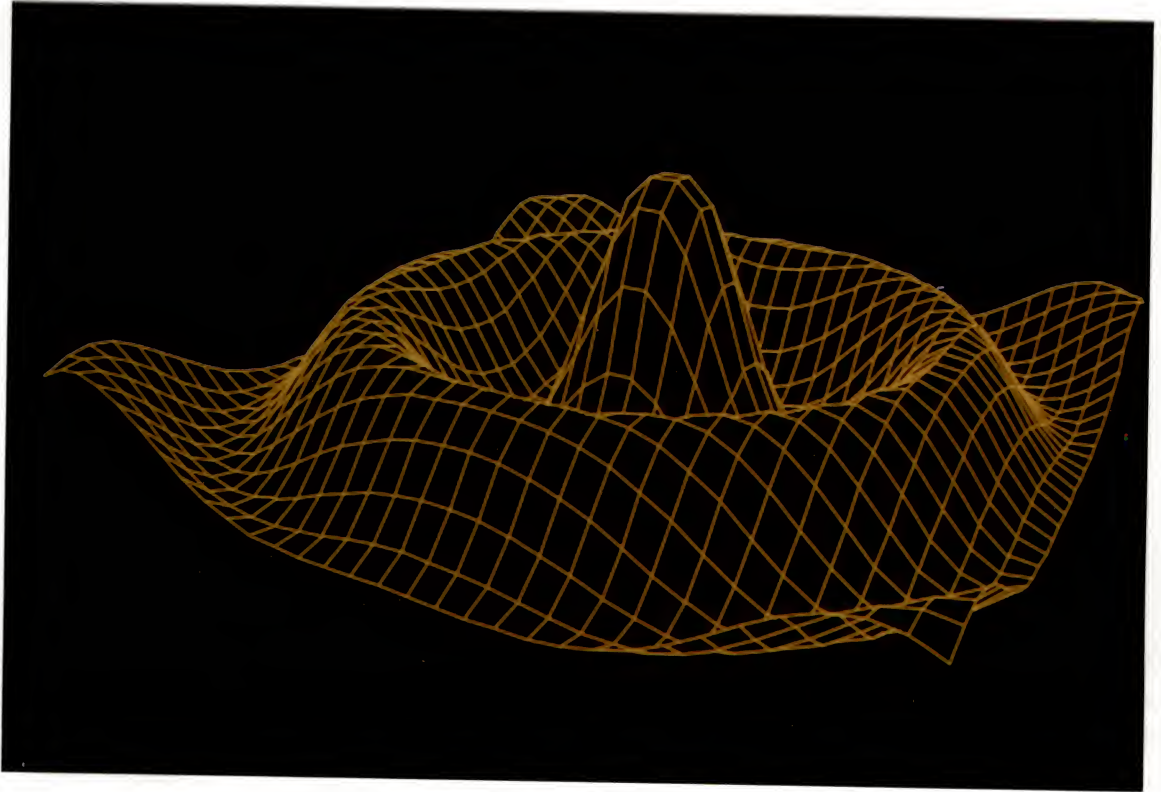


Figure 5-4. A post-anti-aliased gridded surface image of a Bessel function of the first kind.

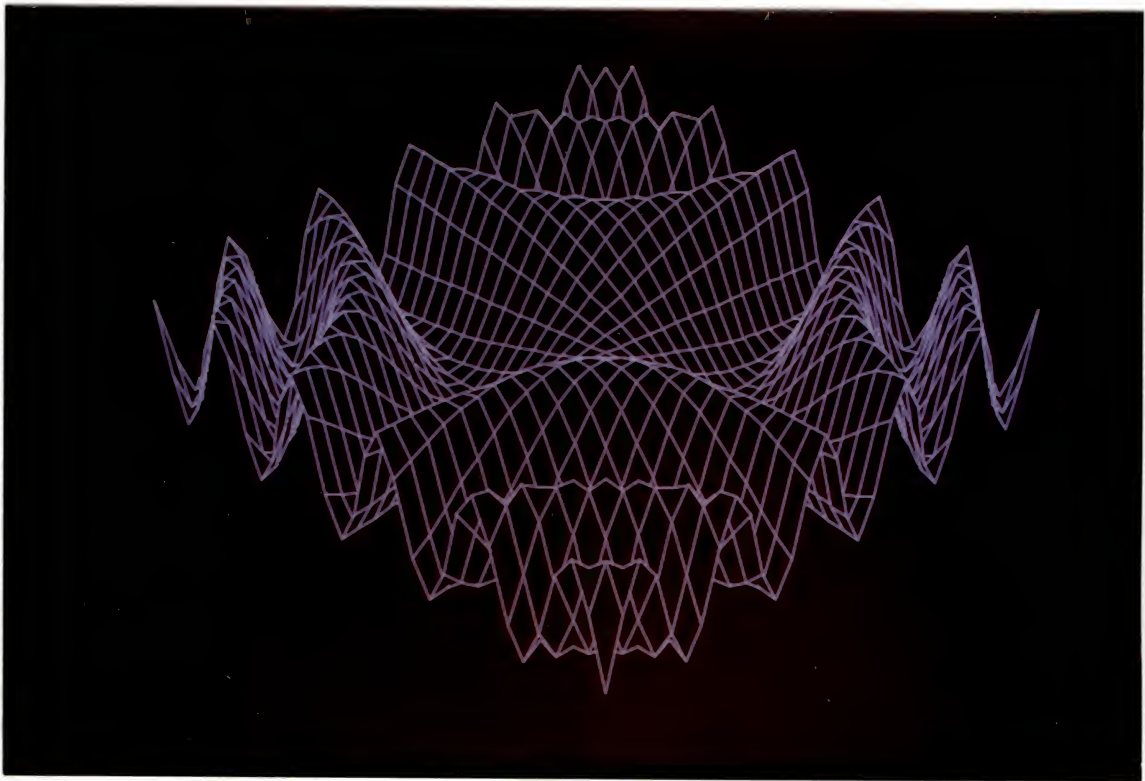
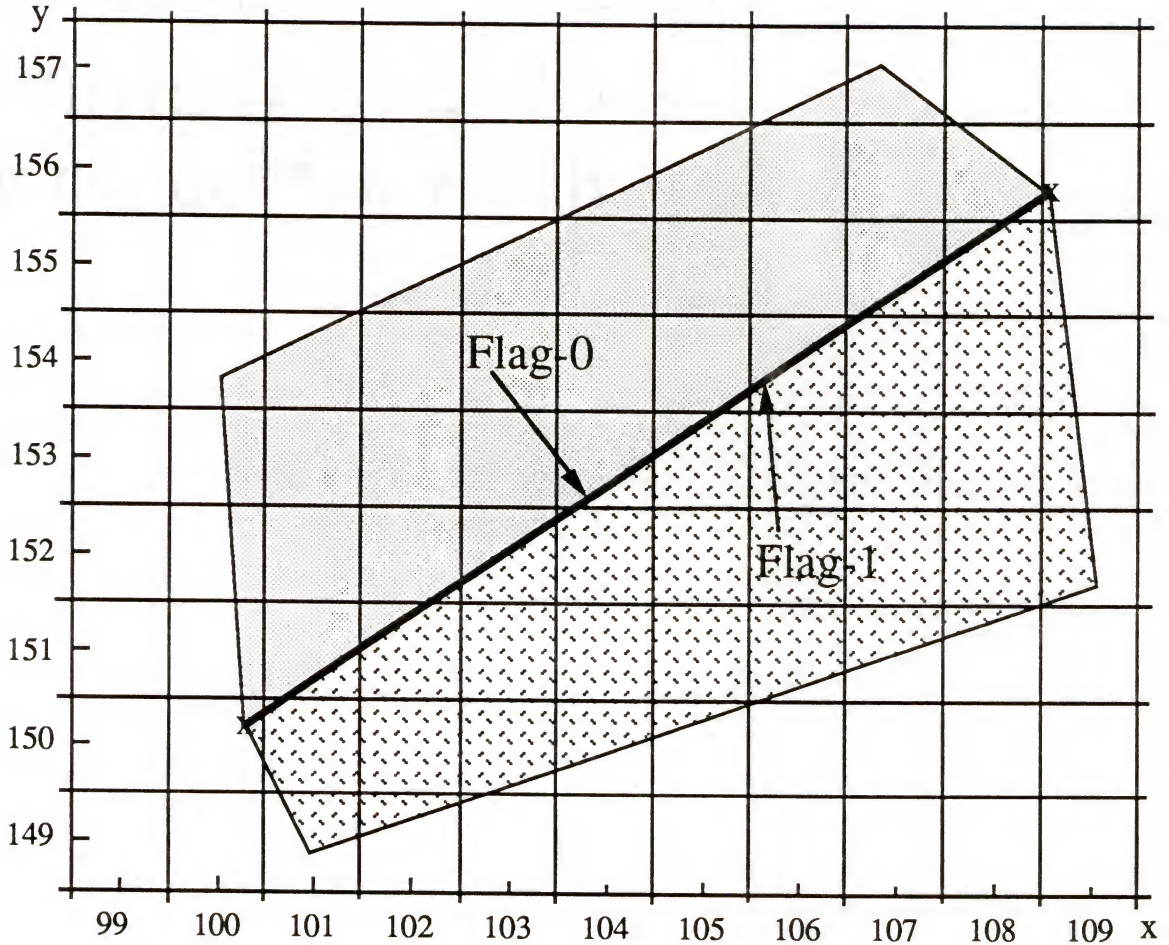


Figure 5-5. A post-anti-aliased gridded surface image of a sine wave function.



(a) Flag- 0

x	y	Area
101	151	0.807524
101	150	0.020192
102	152	0.974819
102	151	0.178196
103	152	0.478315
104	153	0.788889
104	152	0.014726
105	154	0.967804
105	153	0.161111
106	154	0.454216
107	155	0.769395
107	154	0.010121
108	156	0.959929
108	155	0.144887

(b) Flag- 1

x	y	Area
101	151	0.192476
101	150	0.979808
102	152	0.025181
102	151	0.821804
103	152	0.521685
104	153	0.211111
104	152	0.985274
105	154	0.032196
105	153	0.838889
106	154	0.545784
107	155	0.230605
107	154	0.989879
108	156	0.040071
108	155	0.855113

Figure 5-6. Using the algorithm DISLOP_E to anti-alias a common edge with two ends (100.3,150.2) and (108.6,155.8) (a) on the bottom of a polygon (flag- 0) and (b) on the top of the other (flag- 1).

Table 5-3. The area/intensity sampling look-up tables for post-anti-aliasing a polygon edge with flag 0 and a slope of 0.680 at 64 intensity levels.

Slope m => i		Distance dxy => j		Pixel Area Coverage			64 Intensities		
				Upper	Middle	Lower	Up	Mid	Low
0.680	34	-0.500	0	0.0000	0.9150	0.0850	0	58	5
0.680	34	-0.480	1	0.0000	0.9047	0.0753	0	57	5
0.680	34	-0.460	2	0.0000	0.8938	0.0662	0	56	4
0.680	34	-0.440	3	0.0000	0.8824	0.0576	0	56	4
0.680	34	-0.420	4	0.0000	0.8703	0.0497	0	55	3
0.680	34	-0.400	5	0.0000	0.8576	0.0424	0	54	3
0.680	34	-0.380	6	0.0000	0.8444	0.0356	0	53	2
0.680	34	-0.360	7	0.0000	0.8306	0.0294	0	52	2
0.680	34	-0.340	8	0.0000	0.8162	0.0238	0	51	2
0.680	34	-0.320	9	0.0000	0.8012	0.0188	0	50	1
0.680	34	-0.300	10	0.0000	0.7856	0.0144	0	49	1
0.680	34	-0.280	11	0.0000	0.7694	0.0106	0	48	1
0.680	34	-0.260	12	0.0000	0.7526	0.0074	0	47	0
0.680	34	-0.240	13	0.0000	0.7353	0.0047	0	46	0
0.680	34	-0.220	14	0.0000	0.7174	0.0026	0	45	0
0.680	34	-0.200	15	0.0000	0.6988	0.0012	0	44	0
0.680	34	-0.180	16	0.0000	0.6797	0.0003	0	43	0
0.680	34	-0.160	17	0.0000	0.6600	0.0000	0	42	0
0.680	34	-0.140	18	0.0000	0.6400	0.0000	0	40	0
0.680	34	-0.120	19	0.0000	0.6200	0.0000	0	39	0
0.680	34	-0.100	20	0.0000	0.6000	0.0000	0	38	0
0.680	34	-0.080	21	0.0000	0.5800	0.0000	0	37	0
0.680	34	-0.060	22	0.0000	0.5600	0.0000	0	35	0
0.680	34	-0.040	23	0.0000	0.5400	0.0000	0	34	0
0.680	34	-0.020	24	0.0000	0.5200	0.0000	0	33	0
0.680	34	0.000	25	0.0000	0.5000	0.0000	0	31	0
0.680	34	0.020	26	0.0000	0.4800	0.0000	0	30	0
0.680	34	0.040	27	0.0000	0.4600	0.0000	0	29	0
0.680	34	0.060	28	0.0000	0.4400	0.0000	0	28	0
0.680	34	0.080	29	0.0000	0.4200	0.0000	0	26	0
0.680	34	0.100	30	0.0000	0.4000	0.0000	0	25	0
0.680	34	0.120	31	0.0000	0.3800	0.0000	0	24	0
0.680	34	0.140	32	0.0000	0.3600	0.0000	0	23	0
0.680	34	0.160	33	0.0000	0.3400	0.0000	0	21	0
0.680	34	0.180	34	0.9997	0.3203	0.0000	63	20	0
0.680	34	0.200	35	0.9988	0.3012	0.0000	63	19	0
0.680	34	0.220	36	0.9974	0.2826	0.0000	63	18	0
0.680	34	0.240	37	0.9953	0.2647	0.0000	63	17	0
0.680	34	0.260	38	0.9926	0.2474	0.0000	63	16	0
0.680	34	0.280	39	0.9894	0.2306	0.0000	62	15	0
0.680	34	0.300	40	0.9856	0.2144	0.0000	62	14	0
0.680	34	0.320	41	0.9812	0.1988	0.0000	62	13	0
0.680	34	0.340	42	0.9762	0.1838	0.0000	61	12	0
0.680	34	0.360	43	0.9706	0.1694	0.0000	61	11	0
0.680	34	0.380	44	0.9644	0.1556	0.0000	61	10	0
0.680	34	0.400	45	0.9576	0.1424	0.0000	60	9	0
0.680	34	0.420	46	0.9503	0.1297	0.0000	60	8	0
0.680	34	0.440	47	0.9424	0.1176	0.0000	59	7	0
0.680	34	0.460	48	0.9338	0.1062	0.0000	59	7	0
0.680	34	0.480	49	0.9247	0.0953	0.0000	58	6	0
0.680	34	0.500	50	0.9150	0.0850	0.0000	58	5	0

Table 5-4. The area/intensity sampling look-up tables for post-anti-aliasing a polygon edge with flag 1 and a slope of 0.680 at 64 intensity levels.

Slope m => i		Distance dxy => j		Pixel Area Coverage			64 Intensities		
				Upper	Middle	Lower	Up	Mid	Low
0.680	34	-0.500	0	0.0000	0.0850	0.9150	0	5	58
0.680	34	-0.480	1	0.0000	0.0953	0.9247	0	6	58
0.680	34	-0.460	2	0.0000	0.1062	0.9338	0	7	59
0.680	34	-0.440	3	0.0000	0.1176	0.9424	0	7	59
0.680	34	-0.420	4	0.0000	0.1297	0.9503	0	8	60
0.680	34	-0.400	5	0.0000	0.1424	0.9576	0	9	60
0.680	34	-0.380	6	0.0000	0.1556	0.9644	0	10	61
0.680	34	-0.360	7	0.0000	0.1694	0.9706	0	11	61
0.680	34	-0.340	8	0.0000	0.1838	0.9762	0	12	61
0.680	34	-0.320	9	0.0000	0.1988	0.9812	0	13	62
0.680	34	-0.300	10	0.0000	0.2144	0.9856	0	14	62
0.680	34	-0.280	11	0.0000	0.2306	0.9894	0	15	62
0.680	34	-0.260	12	0.0000	0.2474	0.9926	0	16	63
0.680	34	-0.240	13	0.0000	0.2647	0.9953	0	17	63
0.680	34	-0.220	14	0.0000	0.2826	0.9974	0	18	63
0.680	34	-0.200	15	0.0000	0.3012	0.9988	0	19	63
0.680	34	-0.180	16	0.0000	0.3203	0.9997	0	20	63
0.680	34	-0.160	17	0.0000	0.3400	0.0000	0	21	0
0.680	34	-0.140	18	0.0000	0.3600	0.0000	0	23	0
0.680	34	-0.120	19	0.0000	0.3800	0.0000	0	24	0
0.680	34	-0.100	20	0.0000	0.4000	0.0000	0	25	0
0.680	34	-0.080	21	0.0000	0.4200	0.0000	0	26	0
0.680	34	-0.060	22	0.0000	0.4400	0.0000	0	28	0
0.680	34	-0.040	23	0.0000	0.4600	0.0000	0	29	0
0.680	34	-0.020	24	0.0000	0.4800	0.0000	0	30	0
0.680	34	0.000	25	0.0000	0.5000	0.0000	0	32	0
0.680	34	0.020	26	0.0000	0.5200	0.0000	0	33	0
0.680	34	0.040	27	0.0000	0.5400	0.0000	0	34	0
0.680	34	0.060	28	0.0000	0.5600	0.0000	0	35	0
0.680	34	0.080	29	0.0000	0.5800	0.0000	0	37	0
0.680	34	0.100	30	0.0000	0.6000	0.0000	0	38	0
0.680	34	0.120	31	0.0000	0.6200	0.0000	0	39	0
0.680	34	0.140	32	0.0000	0.6400	0.0000	0	40	0
0.680	34	0.160	33	0.0000	0.6600	0.0000	0	42	0
0.680	34	0.180	34	0.0003	0.6797	0.0000	0	43	0
0.680	34	0.200	35	0.0012	0.6988	0.0000	0	44	0
0.680	34	0.220	36	0.0026	0.7174	0.0000	0	45	0
0.680	34	0.240	37	0.0047	0.7353	0.0000	0	46	0
0.680	34	0.260	38	0.0074	0.7526	0.0000	0	47	0
0.680	34	0.280	39	0.0106	0.7694	0.0000	1	48	0
0.680	34	0.300	40	0.0144	0.7856	0.0000	1	49	0
0.680	34	0.320	41	0.0188	0.8012	0.0000	1	50	0
0.680	34	0.340	42	0.0238	0.8162	0.0000	2	51	0
0.680	34	0.360	43	0.0294	0.8306	0.0000	2	52	0
0.680	34	0.380	44	0.0356	0.8444	0.0000	2	53	0
0.680	34	0.400	45	0.0424	0.8576	0.0000	3	54	0
0.680	34	0.420	46	0.0497	0.8703	0.0000	3	55	0
0.680	34	0.440	47	0.0576	0.8824	0.0000	4	56	0
0.680	34	0.460	48	0.0662	0.8938	0.0000	4	56	0
0.680	34	0.480	49	0.0753	0.9047	0.0000	5	57	0
0.680	34	0.500	50	0.0850	0.9150	0.0000	5	58	0

Depending on the object image that is color filled or shaded, we can select area or intensity sampling look-up tables for anti-aliasing its silhouette edges, respectively. A color filled simple object image (see Figure 4-13) has the same intensity value for all pixels. Thus this intensity value is the maximum amount of intensity on silhouette edges and then a set of anti-aliasing intensity values are generated from this maximum value. In this case, using the edge intensity sampling look-up tables to smooth the object silhouette is a better choice since it does the job in a simpler and quicker way.

On the other hand, a color shaded surface image (see Figure 3-4) usually adopts a shading model which takes into account the direction and geometry of the light sources, the surface orientation and the surface properties of the object. Hence a series of different intensity values of three color elements (red, green, blue) are calculated for shading and high-lighting the surface image. In other words, each of the pixels representing the object on a raster display may have a different intensity value including pixels on the silhouette edges. Therefore, to smooth the jagged silhouette of the object image, we must obtain the areas of all pixels, which are passed by silhouette edges. That is, the area sampling look-up tables are selected for post-anti-aliasing silhouette edges of the shaded images.

Before the silhouette of a surface image is smoothed, the true silhouette must be detected. An easy case is shown in Figure 3-4. The silhouette of this image is exactly all around the surface with a jagged margin between the representation of object and the plain background on the display. Since the silhouette is composed of a set of polygon edges, it is clear that, first of all, each edge has to be assigned a flag according to its position related to the polygon, and then the edge is post-anti-aliased with its desired area sampling look-up table and the shaded colors of all passed pixels. As a result, Figure 5-7 and Figure 5-8 show the post-anti-aliased surface images of Figure 3-4 and Figure 3-5, respectively: they have different surface properties and different light source locations. It is noted that only the silhouettes are smoothed here.

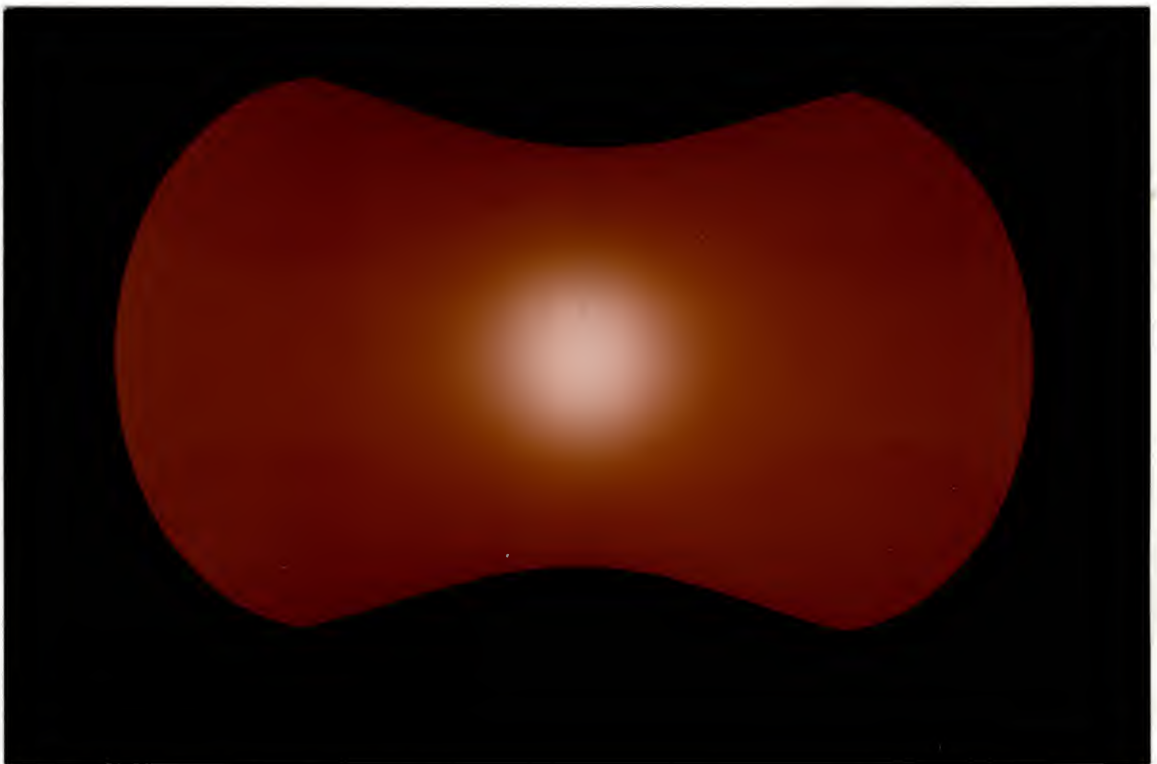


Figure 5-7. A post-anti-aliased image of Figure 3-4 with different surface properties.

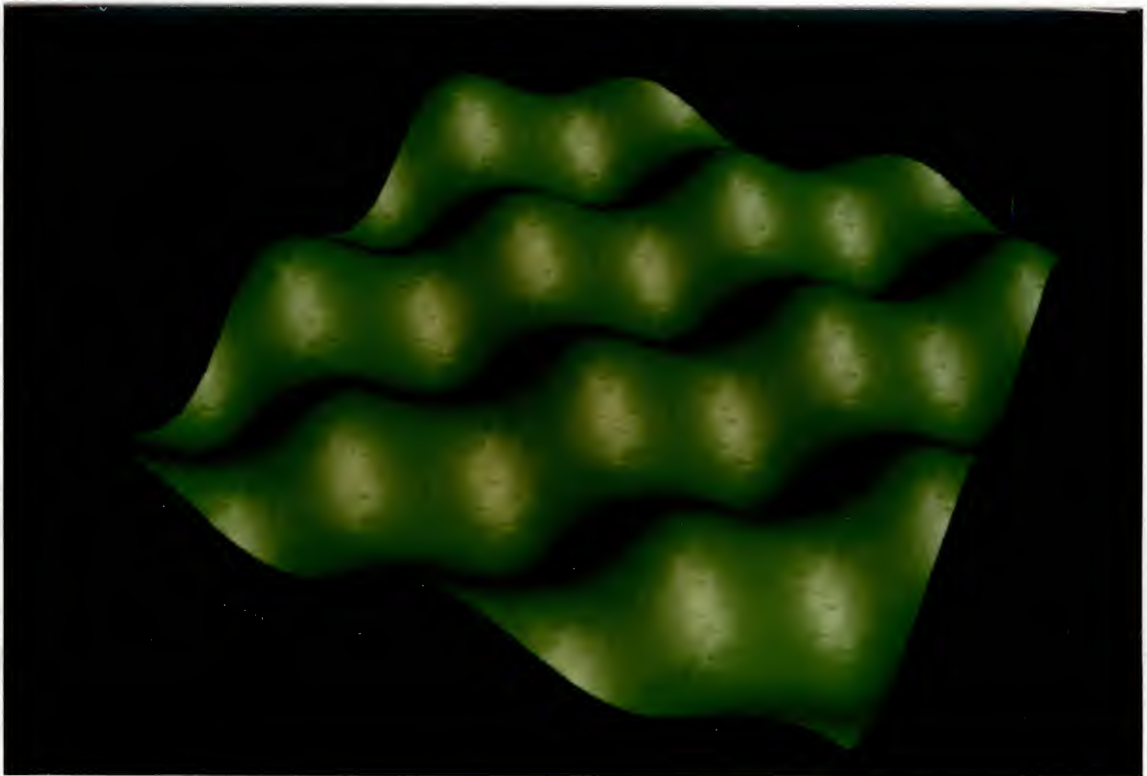


Figure 5-8. A post-anti-aliased image of Figure 3-5 with different surface properties.

Another more complicated case can be observed in Figure 5-9, in which the representation of an object image overlaps itself on a raster display and produces jagged boundaries in between. Therefore, the detection of the true silhouette in the image should be done before smoothing it. A simple way, which usually works quite well, is to compare if there is a big change in the intensity values of adjacent pixels. When the intensity values of two adjacent pixels have much difference often a silhouette edge exists across either one or both of these two pixels. Then the pixels are divided by the passing edge and have to be considered with their corresponding shaded colors in order to do anti-aliasing properly. It is very similar to smoothing an object image on the non-black background so that both colors of the object and the background have to be dealt with. Here, a well-smoothed surface for a sine wave function is calculated and shown in Figure 5-10, which can be used to compare with its non-anti-aliased shaded image in Figure 5-9.

Due to the limitation of the maximum number of colors that can be displayed, or color resolution, on any chosen computer graphics system, the image quality of smoothed shading and anti-aliasing have to be compromised. We use a system with a color resolution of 256, which sometimes may not be enough for displaying a very smooth shaded image with anti-aliased silhouettes. Subjective tests need be done in order to obtain the best image that the system can perform. That is, although the number of shading intensity levels and the number of anti-aliasing intensity levels are inversely related, they are independently selected for the calculation of the intensity values for the color triplets (red, green, blue) of each pixel representing the object in order to get the best outcome without exceeding the number of colors available on the system. All smoothed shaded pictures with smoothed silhouettes in this dissertation were produced in this way.

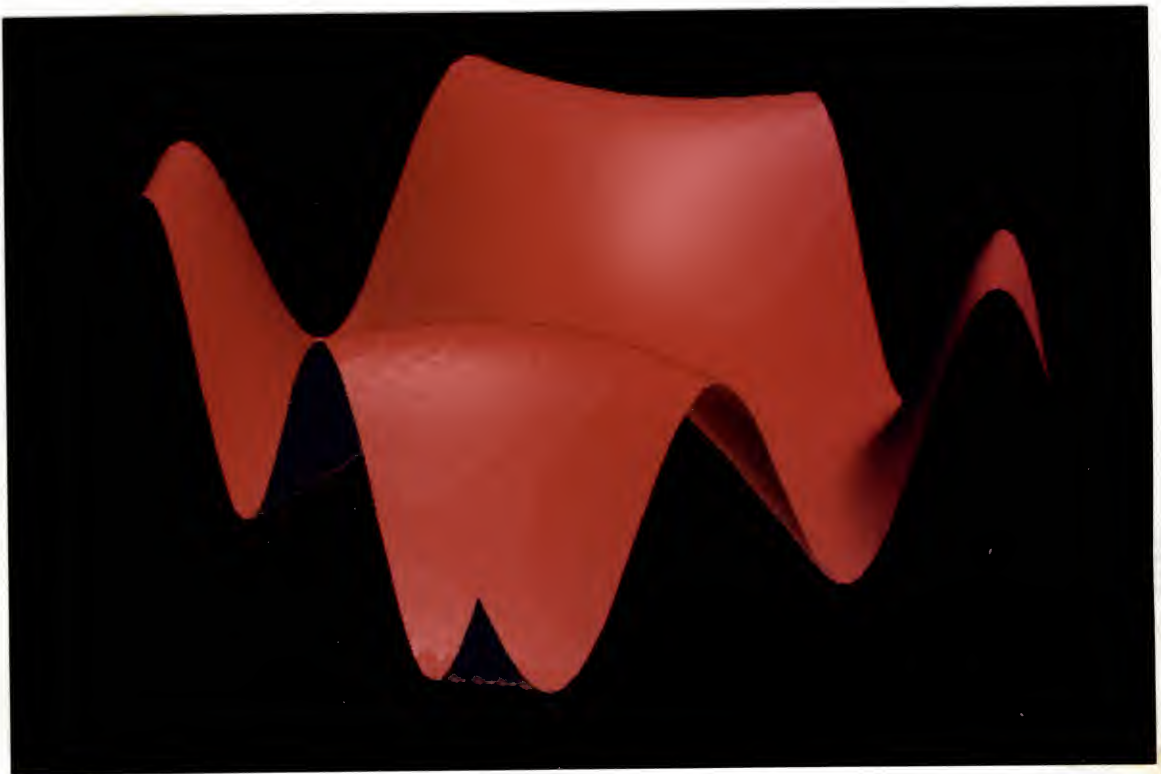


Figure 5-9. A shaded surface image of a sine wave function with the jagged silhouettes.

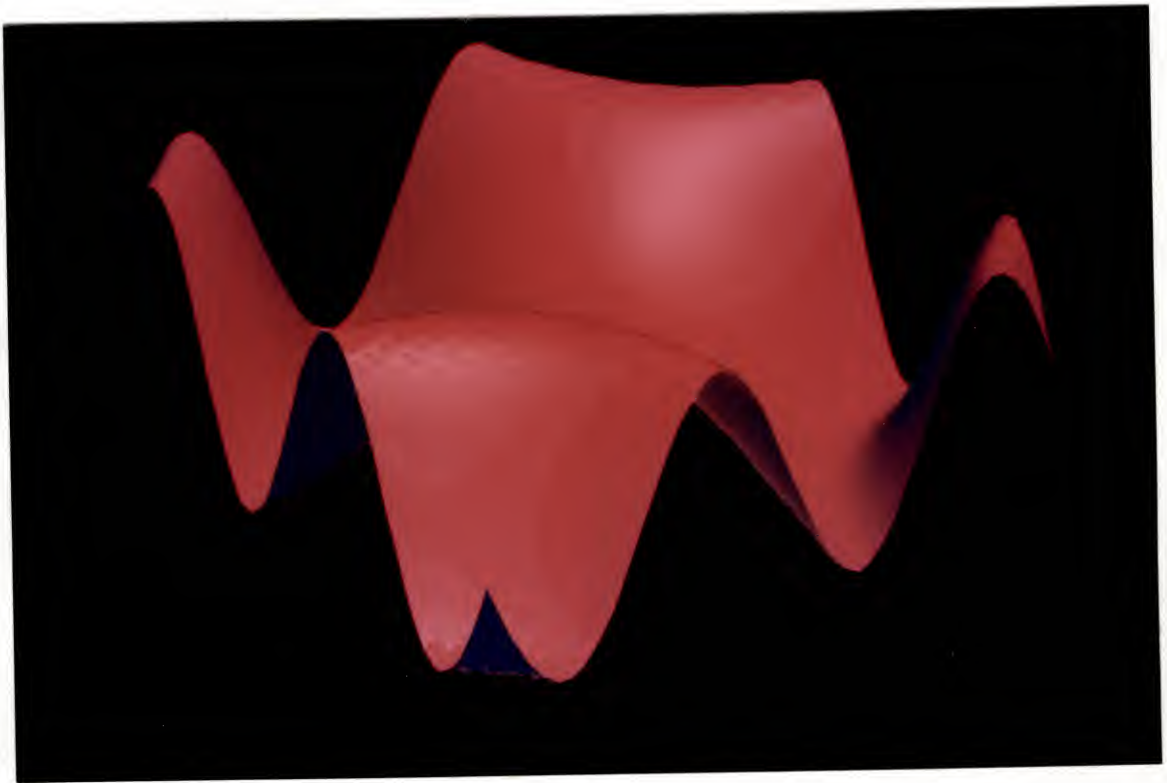


Figure 5-10. A post-anti-aliased image of Figure 5-9.

CHAPTER 6 CONCLUSIONS

This dissertation is primarily concerned with image rendering of objects defined by partially ordered surface data. Algorithms for rapid visibility determination of objects given by algebraic surface functions and for displaying the smoothed realistic images of the objects' visible parts on an interactive raster device are the main results of this research. The resultant procedures are simple, computationally regular and inexpensive; they are in the form necessary for implementation in simple hardware. They are detailed in the previous chapters and are briefly summarized here.

The Floating Perimeter Algorithm offers an efficient way to determine the visibility of object surfaces with partially ordered data by taking advantage of the objects' geometric properties and their projections. Other existing algorithms, such as the Floating Horizon Algorithm and Anderson's algorithm, developed for this kind of objects, have been compared with the FPA with regard to their performance results as well as their implementation requirements. The comparisons show that the FPA is more general, simpler and faster. The FPA is capable of rapidly solving the problem of hidden-line/surface elimination for the image generation of objects given by partially ordered surface data, and it may be easily extended to surfaces with arbitrary ordered data.

In recent years alternate shading techniques have been widely developed, including ray tracing for determining the global illumination for considering interreflections among objects. However, one of the three basic shading techniques, Phong shading, is still widely used since it can generate excellent images without undue computations. This dissertation discusses a modified fast Phong shading that uses a formulation of Taylor's series to approximate Phong's reflection equation which results in considerable reductions of the

amount of computation compared to other existing formulations. Therefore, the method is a candidate for hardware implementation for real-time applications or may be used as software to speed image generation for a variety of display systems.

The image on a raster display is a discrete representation of continuous objects in geometric space. If the objects are sampled at regularly spaced locations in order to be displayed on a raster device, the spacing of the samples limits the amount of detail that can be represented. Consequently, this sampling process causes aliasing artifacts which are most noticeable as jagged image silhouettes.

This dissertation is also concerned with simple methods for producing good-looking images on raster displays. By considering the pixels as abutting squares for the raster structure of the display system, an analytic anti-aliased approach has been developed for removing the aliasing effects from raster images of lines, polygon edges, circles and sphere silhouettes. The algorithms use an area sampling process for each pixel that needs to be anti-aliased, calculate the partial pixel areas covered by the object and then compute the light intensities in each of the pixel parts. The descriptions and pictures shown in the last two chapters show that the anti-aliasing algorithms of line DISLOP_L and edge DISLOP_E can, in general, be applied for smooth rendering of any kind of object. Also a close examination of some published anti-aliasing algorithms, such as Fujimoto's algorithm and recursive sampling, reveals that lines and curves they produce do not have the property of constant-intensity, which is indeed the main requirement of anti-aliasing. However, our anti-aliasing algorithm DISLOP_L does yield constant-intensity lines and curves.

Whenever the speed of image generation as well as image quality are concerns, post-anti-aliasing a final image by means of line or edge area/intensity sampling look-up tables should be a computationally acceptable choice. These tables are generated based on the periodicity of repeating elements in sections of lines or edges determined by their slopes and the anti-aliasing algorithm DISLOP_L or DISLOP_E, respectively. Hence, all numbers in the tables are, in fact, calculated from the above area sampling algorithms and then stored

for use during post-anti-aliasing. The size of tables is set as $51 \times 51 \times 3$ for anti-aliasing 51 sets of 51 lines (edges)--all lines (edges) in one set have the same slope in the first range but have different distances with the center of a passed pixel-- with the maximum number of covered pixels per column, 3, at 64 intensity levels. The size of tables can be changed at the user's pleasure, especially when the number of anti-aliasing intensity levels needs to be changed, such as 32 or 16. Also both the line and edge area/intensity sampling look-up tables could be used to make anti-aliasing post-processors for providing fast and proper anti-aliasing features on raster display devices.

By combining the above algorithms of visibility determination, shading and anti-aliasing, two extended algorithms are created for the image generation of objects given by partially ordered surface data and described in Chapter 5: a hidden-line algorithm with anti-aliasing/post-anti-aliasing and a smoothed visible surface algorithm. The implementation of either one of the two algorithms can represent objects by visible smoothed lines, or shaded surfaces and smoothed silhouettes, on a raster display. Comparisons between the algorithms using anti-aliasing and post-anti-aliasing approaches show that the quality of images produced is almost indistinguishable. However, there are differences in the computational complexity. The area/intensity sampling look-up tables should be utilized for quickly post-anti-aliasing otherwise fully rendered images.

This dissertation provides a novel, complete software system for generating high quality realistic images of solid objects defined by partially ordered surface data on a raster display. Also, the smoothed visible surface algorithm could be used with some existing [Naka89] or to-be-developed techniques to compose smoothed shaded surfaces and manage their databases for the fast generation of realistic complex scenes.

BIBLIOGRAPHY

- Ande82 Anderson, D.P., "Hidden Line Elimination in Projected Grid Surfaces," ACM Transactions on Graphics, Vol. 1, No. 4, October 1982, pp. 274-288.
- Appe67 Appel, A., "The Notion of Quantitative Invisibility and Machine Rendering of Solids," Proc. ACM National Conference, 1967, pp. 387-393.
- Athe83 Atherton, P. R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 73-82.
- Bish86 Bishop, G. and Weimer, D.M., "Fast Phong Shading," Computer Graphics, Vol. 20, No. 4, August 1986, pp. 103-106.
- Blin77 Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures," Computer Graphics, Vol. 11, No. 2, July 1977, pp. 192-198.
- Blin78 Blinn, J.F., "A Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," Computer Graphics, Vol. 12, No. 3, August 1978, pp. 27.
- Blin82 Blinn, J.F., "A Generalization of Algebraic Surfaces Drawing," ACM Transaction on Graphics, Vol. 1, No. 3, July 1982, pp. 235-256.
- Blin88 Blinn, J.F., "Fractional Invisibility," IEEE Computer Graphics and Application, Vol. 8, No. 6, November 1988, pp. 77-84.
- Bloo83 Bloomenthal, J., "Edge Inference with Applications to Antialiasing," Computer Graphics, Vol. 17, No.3, July 1983, pp. 157-162.
- Bouk69 Bouknight, W.J., "An Improved Procedure for Generation of Half-tone Computer Graphics Representations," University of Illinois, Coordinated Science Laboratory, R-432, September 1969.
- BuiT75 Bui-Tuong, Phong, "Illumination for Computer Generated Pictures," CACM, Vol. 18, No. 6, June 1975, pp. 311-317
- Butl79 Butland, J., "Surface Drawing Made Simple," Computer Aided Design, Vol. 11, No. 1, January 1979, pp 19-22.
- Carp84 Carpenter, L., "The A-buffer, an Antialiased Hidden Surface Method," Computer Graphics, Vol. 18, No. 3, July 1984, pp. 103-108.
- Catm74 Catmull, E., A Subdivision Algorithm for Computer Display of Curved Surfaces, Ph.D. Dissertation, University of Utah, Salt Lake City, Utah, 1974.

- Catm75 Catmull, E., "Computer Display of Curved Surfaces," Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures, May 1975, pp.11-17.
- Catm78 Catmull, E., "A Hidden-surface Algorithm with Anti-aliasing," Computer Graphics, Vol. 12, No. 3, August 1978, pp. 6-11.
- Catm79 Catmull, E., "A Tutorial on Compensation Tables," Computer Graphics, Vol. 13, No. 2, August 1979, pp. 1-7.
- Catm84 Catmull, E., "An Analytic Visible Surface Algorithm for Independent Pixel Processing," Computer Graphics, Vol. 18, No. 3, July 1984, pp.109-115.
- Chen83 Chen, K.T., Design of A Run-Length Decoder, MS Thesis, Department of Electrical Engineering, University of Florida, 1983.
- Cohe85 Cohen, M.F. and Greenberg, D.P., "The Hemi-Cube: A Radiosity Solution for Complex Environment," Computer Graphics, Vol. 19, No. 3, July 1985, pp. 31-40.
- Conr85 Conrac Division, Conrac Corporation, Raster Graphics Handbook, Second Edition, Van Nostrand Reinhold Company Inc., New York, 1985.
- Cook86 Cook, R.L., "Stochastic Sampling in Computer Graphics," ACM Transactions on Graphics, Vol. 5, No. 1, January 1986, pp. 51-72.
- Cook87 Cook, R.L., Carpenter, L. and Catmull, E., "The Reyes Image Rendering Architecture," Computer Graphics, Vol. 21, No. 4, July 1987, pp. 95-102.
- Cowa83 Cowan, W.B., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIE Standard Coordinates," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 315-321.
- Cowa85 Cowan, W.B. and Ware, C., Color Perception, Course Notes #3, ACM SIGGRAPH'85, July 1985, pp. 44-53.
- Croc84 Crocker, G.A., "Invisibility Coherence for Faster Scan-Line Hidden Surface Algorithms," Computer Graphics, Vol. 18, No. 3, July 1984, pp. 95-102.
- Crow77 Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images," CACM, Vol. 20, No. 11, November 1977, pp. 799-805.
- Crow81a Crow, F.C., "A Comparison of Antialiasing Techniques," IEEE Computer Graphics and Applications, Vol. 1, No. 1, January 1981, pp. 40-48.
- Crow81b Crow, F.C., "Three-Dimensional Computer Graphics, Part 1," BYTE, March 1981, pp. 54-82.
- Crow81c Crow, F.C., "Three-Dimensional Computer Graphics, Part 2," BYTE, April 1981, pp. 290-302.
- Dres84 Dresdner, D., High-Speed Communication with An Intelligent Frame Buffer, MS Thesis, Department of Electrical Engineering, University of Florida, 1984.

- Duff79 Duff, Tom, "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays," *Computer Graphics*, Vol. 13, No. 2, August 1979, pp. 270-275.
- Dyer87 Dyer, S. and Whitman, S., "A Vectorized Scan-Line Z-Buffer Rendering Algorithm," *IEEE Computer Graphics and Applications*, Vol. 7, No. 7, 1987, pp. 34-44.
- Feib80 Feibush, E.A., Levoy, M. and Cook, R.L., "Synthetic Texturing Using Digital Filters," *Computer Graphics*, Vol. 14, No. 3, 1980, pp. 294-301.
- Ferw88 Ferwerda, J.A. and Greenberg, D.P., "A Psychophysical Approach to Assessing the Quality of Antialiased Images," *IEEE Computer Graphics and Applications*, Vol. 8, No. 5, September 1988, pp. 85-95.
- Fole82 Foley, J.D. and VanDam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1982.
- Fren89 Frenkel, K.A., "An Interview with Ivan Sutherland," *CACM*, Vol. 32, No. 6, June 1989, pp. 711-718.
- Frie85 Frieder, G., Gorden, D. and Reynolds, R.A., "Back-to-Front Display of Voxel Based Objects," *IEEE Computer Graphics and Applications*, Vol. 5, No. 1, 1985, pp. 52-60.
- Fuji83 Fujimoto, A. and Iwata, K., "Jag-Free Images on Raster Display," *IEEE Computer Graphics and Applications*, Vol. 3, No. 9, December 1983, pp. 26-34.
- Gali69 Galimberti, R. and Montanari, U., "An Algorithm for Hidden-line Elimination" *CACM*, Vol. 12, No. 4, April 1969, pp. 206-211.
- Gora84 Goral, C.M., Torrance, K.E., and Greenberg, D.P., "Modeling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics*, Vol. 18, No. 3, July 1984, pp. 213-222.
- Gour71 Gouraud, H., "Computer Display of Curved Surfaces," *IEEE Transactions on Computers*, Vol. C-20, No. 6, June 1971, pp. 623-629.
- Gran85 Grant, C.W., "Integrated Analytic Spatial and Temporal Anti-Aliasing for Polyhedra in 4-Space," *Computer Graphics*, Vol. 19, No. 3, July 1985, pp. 79-84.
- Gran87 Grant, C.W., "A Preliminary Taxonomy of Visible Surface Algorithms," *Workshop on Rendering Algorithms and Systems, CHI+GI 87*, Toronto, Ontario, April 1987.
- Ham177 Hamlin, Jr., G. and Gear, C.W., "Raster-Scan Hidden Surface Algorithm Techniques," *Computer Graphics*, Vol. 11, No. 2, 1977, pp. 206-213.
- Hanr83 Hanrahan, P., "Ray Tracing Algebraic Surfaces," *Computer Graphics*, Vol. 17, No. 3, July 1983, pp. 83-90.

- Hedg82 Hedgley, D.R., Jr., "A General Solution to the Hidden-Line Problem," NASA Reference Publication 1085, 1982.
- Hern86 Hearn, D. and Baker, M.P., Computer Graphics, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
- Hunt75 Hunt, R.W.G., The Reproduction of Color, 3rd ed., Wiley, New York, 1975.
- Jen84 Jen, R.T., Anti-aliasing Spherical Silhouette Images for Computer Graphics, MS Thesis, Department of Computer and Information Sciences, University of Florida, 1984.
- Joy88 Joy, K.I., Grant, C.W., Max, N.L. and Hatfield, L., Computer Graphics: Image Synthesis, Computer Society Press, Washington D.C., 1988.
- Kaji81 Kajiya, J. and Uller, M., "Filtering High Quality Text for Display on Raster Scan Device," Computer Graphics, Vol. 15, No. 3, August 1981, pp. 7-15.
- Khur84 Khurana, A.S., Molecular Modelling in Real-Time, MS Thesis, Department of Electrical Engineering, University of Florida, 1984.
- Knut73 Knuth, D.E., The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Massachusetts, 1973.
- Kore83 Korein, J. and Badler, N.I., "Temporal Anti-Aliasing in Computer Generated Animation," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 377-388.
- Lane80 Lane, J.M., Carpenter, L.C., Whitted, T. and Blinn, J.F., "Scan Line Methods for Displaying Parametrically Defined Surfaces," CACM, Vol. 23, No. 1, January 1980, pp. 23-34.
- Lee85 Lee, M.E., Redner, R.A. and Uselton, S.P., "Statistically Optimized Sampling for Distributed Ray Tracing," Computer Graphics, Vol. 19, No. 3, July 1985, pp. 61-67.
- Lout67 Loutrel, P.P., "A Solution to the Hidden-line Problem for Computer-drawn Polyhedra," Department of Electrical Engineering, New York, Technical Report 400-167, September 1967.
- Max85 Max, N.L. and Lerner, D.M., "A Two-and-a-Half Motion-Blur Algorithm," Computer Graphics, Vol. 19, No. 3, July 1985, pp. 85-93.
- McAl81 McAllister, D.F. and Roulier, J.A., "An Algorithm for Computing a Shape Preserving Osculatory Quadratic Spline," ACM Trans. Mathematical Software, Vol. 7, No. 3, September 1981, pp. 331-347.
- McKe87 McKenna, M., "Worst-Case Optimal Hidden-Surface Removal," ACM Transactions on Graphics, 6, 1, January 1987, pp. 19-28.
- Meye88 Meyer, G.W. and Greenberg, D.P., "Color-Defective Vision and Computer Graphics Displays," IEEE Computer Graphics and Applications, Vol. 8, No. 5, September 1988, pp. 28-40.

- Mont87 Montani, C. and Re, M., "Vector and Raster Hidden-Surface Removal Using Parallel Connected Stripes," IEEE Computer Graphics and Applications, Vol. 7, No. 7, July 1987, pp. 14-23.
- Myer75 Myers, A.J., "An Efficient Visible Surface Program," Report to the NSF, Ohio State University Computer Graphics Research Group, July 1975.
- Naka89 Nakamae, E., Ishizaki, T., Nishita, T., and Takita, S., "Compositing 3D Images with Anti-aliasing and Various Shading Effects," IEEE Computer Graphics and Applications, Vol. 9, No. 2, March 1989, pp. 21-29.
- Newe72 Newell, M.E., R.G. Newell, and T.L. Sancha, "A New Approach to the Shaded Picture Problem," Proceedings of the ACM National Conference, 1972, pp. 443.
- Newm79 Newman, W.M. and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1979.
- Nish85 Nishita, T. and Nakamae, E., "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," Computer Graphics, Vol. 19, No. 3, July 1985, pp. 23-30.
- Nort82 Norton, A., Rockwood, A.P. and Skolmoski, P.T., "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space," Computer Graphics, Vol. 16, No. 3, July 1982, pp. 1-8.
- Nurm85 Nurmi, O., "A Fast Line-Sweep Algorithm for Hidden Line Elimination," BIT, Vol. 25, No. 3, 1985, pp. 466-472.
- Oppe75 Oppenheim, A.V. and Shafer, R.W., "Digital Signal Processing," Prentice-Hall, Englewood Cliffs, New Jersey, 1975.
- Pitt80 Pitteway, M.L.V. and Watkinson, D.J., "Bresenham's Algorithm with Grey Scale," CACM, Vol. 23, No. 11, November 1980, pp. 625-626.
- Potm83 Potmesil, M. and Chakravarty, I., "Modeling Motion Blur in Computer-generated Images," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 389-399.
- Reev83 Reeves, W.T., "Particle Systems- A Technique for Modeling a Class of Fuzzy Objects," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 359- 376.
- Reev85 Reeves, W.T. and Blau, R., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," Computer Graphics, Vol. 19, No. 3, July 1985, pp. 313-322.
- Robe63 Roberts, L.G., "Machine Perception of Three Dimensional Solids," MIT Lincoln Laboratory, TR 315, May 1963.
- Rock87 Rockwood, A.P., "A Generalized Scanning Technique for Display of Parametrically Defined Surfaces," IEEE Computer Graphics and Applications, Vol. 7, No. 8, August 1987, pp. 15-26.

- Roge85 Rogers, D.F., *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.
- Roul87 Roulier, J.A., "A Convexity-Preserving Grid Refinement Algorithm for Interpolation of Bivariate Functions," *IEEE Computer Graphics and Applications*, Vol. 7, No. 1, January 1987, pp. 57-62.
- Same84 Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 187-260.
- Same88a Samet, H. and Webber, R.E., "Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals," *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May 1988, pp.48-68.
- Same88b Samet, H. and Webber, R.E., "Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Applications," *IEEE Computer Graphics and Applications*, Vol. 8, No. 4, July 1988, pp. 59-75.
- Schu69 Schumacker, R.A., B. Brand, M. Gilliland and W. Sharp, "Study for Applying Computer-generated Images for Visual Simulation," Technical Report, AFHRL-TR-69-14, U.S. Air Force Human Resources Lab., September 1969, NTIS AD 700 375.
- Sede88 Sederberg, T.W., "Techniques for Cubic Algebraic Surfaces," Course Notes #25, ACM SIGGRAPH'88, August, 1988.
- Sieg81 Siegel, R. and Howell, J.R., *Thermal Radiation Heat Transfer*, Hemisphere Publishing Corp., Washington, D.C., 1981.
- Spar78 Sparrow, E.M., *Radiation Heat Transfer*, McGraw-Hill, New York, 1978.
- Stau75 Staudhammer, J., "Multi-Dimensional Function Display Using A Color Scale," *Computer Graphics*, Vol. 9, No. 2, 1975, pp. 181-183.
- Stau78 Staudhammer, J., "Coherence Concepts in Computer Synthesized Real-Time Displays," *Proc. 2nd Army Software Symposium*, October 1978, pp. 562-579.
- Stau87 Staudhammer, J. and Wang, S.C., "Visualization With Dynamic Graphics," *NCGA'87 Proceedings*, March 1987, pp. 628-632.
- Stau88 Staudhammer, J., Livadas, P.E., Wang, S.C. and Zhou, X., "Molecular Structure Visualization and Optimal Surface Construction," *NCGA'88 Proceedings*, March 1988, pp. 168-177.
- Stin85 Stinson, D.R., *An Introduction to the Design and Analysis of Algorithms*, Charles Babbage Research Center, Winnipeg Manitoba, Canada, 1985.
- Stoc72 Stockham, T.G., "Image Processing in the Context of a Visual Model," *Proceedings of the IEEE*, No. 6, July 1972, pp. 829-842.
- Ston88 Stone, M.C., Cowan, W.B. and Beatty, J.C., "Color Gamut Mapping and the Printing of Digital Color Images," *ACM Transactions on Graphics*, Vol. 7, No. 3, October 1988, pp. 249-292.

- Suth74 Sutherland, I.E., Sproull, R.F. and Schumacher, R.A., "A Characterization of Ten Hidden-Surface Algorithms," ACM Computing Surveys, Vol. 6, No. 1, March 1974, pp. 1-55.
- Torr67 Torrance, K.E. and Sparrow, E.M., "Theory for Off-Specular Reflection from Roughened Surfaces," J. Opt. Soc. Am., Vol. 57, No. 9, September 1967, pp. 1105-1114.
- Turk82 Turkowski, K., "Anti-aliasing through the Use of Coordinate Transformations," ACM Transactions on Graphics, Vol. 1, No. 3, July 1982, pp. 215-234.
- Wang84 Wang, S.C., "Contributions to the Theory and Practice of Anti-aliasing of Lines, Edges, Circles and Spheres," Report # 51, Computer Graphics Research Laboratory, University of Florida, May 1984.
- Wang85a Wang, S.C., "Visibility of Projected Grid Surfaces," Report # 52, Computer Graphics Research Laboratory, University of Florida, April 1985.
- Wang85b Wang, S.C., "Algorithm for Multi-Dimensional Function Display Using A Color Scale," Report # 53, Computer Graphics Research Laboratory, University of Florida, September 1985.
- Wang86 Wang, S.C., "The Floating Perimeter Algorithm," Report # 24, Computer Graphics Research Laboratory, University of Florida, January 1986.
- Ward89 Ward, F., "Images for the Computer Age," National Geographic, Vol. 175, No. 6, June 1989, pp. 718-751.
- Ware88 Ware, C., "Color Sequences for Univariate Maps: Theory, Experiments, and Principles," IEEE Computer Graphics and Applications, Vol. 8, No. 5, September 1988, pp. 41-49.
- Warn69 Warnock, J.E., A Hidden-Surface Algorithm for Computer Generated Half-toned Pictures, University of Utah, Computer Science Department, TR 4-15, 1969, NTIS AD-753 671.
- Warn80 Warnock, J.E., "The Display of Characters Using Gray Level Sample Arrays," Computer Graphics, Vol. 14, No. 3, 1980, pp. 302-307.
- Watk70 Watkins, G.S., A Real-Time Visible Surface Algorithm, Computer Science Department, University of Utah, UTECH-CSC-70-101, June 1970.
- Weil77 Weiler, K. and P. Atherton, "Hidden Surface Removal Using Polygon Area Sorting," Computer Graphics, Vol. 11, No. 3, 1977, pp. 214-222.
- Whit78 Whitted, T., "A Scan-line Algorithm for Computer Display of Curved Surfaces," Computer Graphics, Vol. 12, No. 3, August 1978, pp. 26.
- Whit80 Whitted, T., "An Improved Illumination Model for Shaded Display," CACM, Vol. 23, No. 6, June 1980, pp. 343-349.
- Whit83 Whitted, T., "Anti-aliased Line Drawing Using Brush Extrusion," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 151-156.

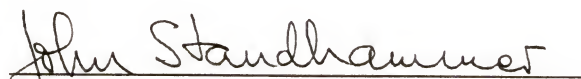
- Will72 Williamson, H., "Hidden-Line Plotting Program," CACM, Vol. 15, No. 2, February 1972, pp. 100-103.
- Will83 Williams, L., "Pyramidal Parametrics," Computer Graphics, Vol. 17, No. 3, July 1983, pp. 1-11.
- Wrig73 Wright, T.J., "A Two-space Solution to the Hidden Line Problems for Plotting Functions of Two Variables," IEEE Transactions on Computers, Vol. C-22, No. 1, January 1973, pp. 28-33.
- Wyli67 Wylie, C., Romney, G.W., Evans, D.C. and Erdahl, A., "Halftone Perspective Drawings by Computer," Proc. AFIPS FJCC, Vol. 31, 1967.

BIOGRAPHICAL SKETCH


Sue-Ling Chen Wang received a BS in chemical engineering from National Cheng Kung University, Taiwan, Republic of China, in 1976. After graduation, she was first employed as engineer and later promoted to specialist in the Patent Office of National Bureau of Standards, Ministry of Economic Affairs, Taiwan, R.O.C. Her duties involved all phases of investigating patent applications of chemical engineering, including classification, procedural and professional technique examinations. She obtained a Certificate of Patent Agent in 1981.

Seeking a higher education, she was admitted to the master's program in computer and information sciences at the University of Florida in the Spring semester of 1982. In addition to her graduate studies, she also worked as a teaching assistant of computer graphics and research assistant at the Interactive Graphics Laboratory, Computer Graphics Research Laboratory, CAD/CAM Facility, Advanced Materials Research Center, and the Department of Nuclear Engineering Sciences since 1983. She received her M.S. in August 1984. With her advisor Dr. Staudhammer's encouragement, she entered the Ph.D. program and then passed the doctoral qualifying examination in the Spring semester, 1988.

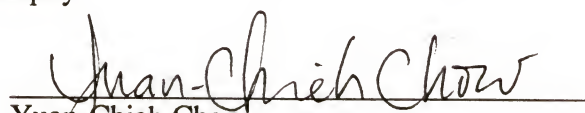
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


John Staudhammer, Chairman
Professor of Computer and Information
Sciences


I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


Joseph Duffy
Graduate Research Professor of Mechanical
Engineering

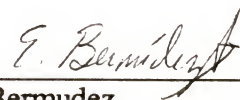
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


Yuan-Chieh Chow
Professor of Computer and Information
Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.


Douglas D. Dankel, II
Assistant Professor of Computer and
Information Sciences

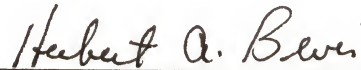
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Manuel E. Bermudez
Assistant Professor of Computer and
Information Sciences

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1989



Dean, College of Engineering

Dean, Graduate School